Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER IN COMPUTER SCIENCE

Consensus algorithm through asymmetric broadcasting in low-complexity sensor networks

Désiron, Kevin

Award date:
2017

Awarding institution:
University of Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. May. 2023

UNIVERSITÉ DE NAMUR
Faculty of Computer Science
Academic Year 2016–2017

Consensus algorithm through asymmetric broadcasting in low-complexity sensor networks

Kevin DÉSIRON

Internship mentor: Ferdinand PEPER, PhD

Supervisor: _____ (Signed for Release Approval - Study Rules art. 40)
Prof. Marie-Ange REMICHE

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Abstract

Consensus algorithm through asymmetric broadcasting in low-complexity sensor networks

by Kevin DÉSIRON

Abstract in English

With the growth of the Internet of Things, the sheer amount of wireless traffic doos not stop to increase and will cause network congestion in a near future. This requires a rethinking of the way we process information, especially in high-density networks. This subject has attracted a lot of interest for the average consensus problem.

In this work, we first provide an overview of actual technology for processing information in sensor network and review the basis of low-complexity sensor and high-density network.

Then, we present the Peper's Average Consensus algorithm in asymmetric broadcasting wireless sensor networks in asynchronous case. We provide a new mathematical model for this algorithm and expose numerical results about the impact of network topology.

After that, we develop a new Average Consensus algorithm with multiple updates as an improvement of the Peper's algorithm and prove its better performance through computer simulations. We also highlight links between some algorithm parameters and give a mathematical model of this algorithm.

Finally, we show a Clustering Consensus algorithm to solve the problem of computing average inside clusters instead of the whole network. We expose our motivations and describe different policies before presenting a numerical evaluation.

Abstract en Français

Avec la croissance de l'Internet des Objets, le traffic sans-fil ne cesse d'augmenter et causera une congestion du réseau dans un avenir proche. Ce problème nous demande de repenser la manière dont nous traitons l'information, surtout dans les réseaux très denses. Cette problématique a poussé beaucoup de chercheurs à s'intéresser au problème de consensus sur la moyenne.

Dans ce travail, nous présentons un état de l'art sur les méthodes actuelles pour traiter l'information dans les réseaux de senseurs et rapellons les bases des senseurs peu complexes et des réseaux très denses.

Ensuite, nous présentons l'algorithme de consensus sur la moyenne de Peper. Nous fournissons un modèle mathématique de celui-ci et analysons l'impact de la topologie du réseau sur l'algorithme.

Après, nous développons une amélioration de cet algorithme en introduisant le concept de mises à jour multiples. Nous analysons ses performances au moyen de simulations et mettons aussi en évidence les liens entre certains paramètres. Nous fournissons aussi un modèle mathématique de cet algorithme.

Finalement, nous présentons un algorithme de consensus sur la moyenne à l'intérieur des clusters présent dans le réseau. Nous exposons nos motivations pour la création d'un tel algorithme et expliquons différentes manières de réaliser cet objectif avant de fournir une évaluation de ces méthodes à partir de simulations.

Acknowledgements

First, I would like to thank my supervisor, Prof Marie-Ange REMICHE, that gave me the opportunity to spend my internship in Japan. This was a great experience and a lot of discovery about a culture I like. I also want to give her my appreciation for the support she gives me in all my administrative procedures and for the exciting subject she proposed.

Second, I thanks Dr Ferdinand PEPER, my internship mentor, for all his support in my works. It was a unique experience to work in a scientific research context with him and his colleague, Dr Kenji LEIBNITZ. They made everything to involve me in their work and they dedicate plenty of time to my supervision. Thanks to them, I discovered the reality of scientific world which was a gold opportunity for me. I hope I can work again with them in the future.

Then, I also want to greet all the help I received from the administrative staff of the Center for Information and Neural Network (CiNet). They efficiently assisted me in my administrative procedures in Japan and they offered me many facilities to make my internship in Japan possible. I wish to express a special thanks to Miss Akiko NAMIKATA that learned me a lot about Japanese culture and history.

Finally, I would like to thank the National Institute of Information and Communications Technology (NICT) for approving and funding my internship.

Contents

Abstract **iii**

Acknowledgements **v**

List of abbreviations **xiii**

Glossary **xv**

Introduction **1**

1 Overview of average algorithms **3**

1.1 Centralized algorithms 3

1.2 Distributed algorithms 4

1.3 Overview of randomized gossiping 5

2 Low-complexity nanosensors netkors **7**

2.1 Limitations on low-complexity nanosensors 7

2.1.1 Energy constraint 7

2.1.2 Computation power and memory constraint 8

2.1.3 Communication range constraint 8

2.1.4 Lack of identification 8

2.2 Impact on network 9

2.2.1 Communication routing 9

2.2.2 Network topology 10

2.3 Effect on average algorithms 11

3 Average consensus algorithm with single update **13**

3.1 Algorithm presentation 13
3.2 Mathematical model 15
3.3 Numerical results 18

4 Average consensus algorithm with multiple updates 21

4.1 Algorithm presentation 21
4.2 Mathematical model 23
4.3 Numerical results 26

5 Clustering consensus algorithm 35

5.1 Motivations 35
5.2 Algorithm presentation 35
5.3 Numerical results 38

Conclusion 45

List of Figures

2.1 Illustration of (a) a Random Geometric Graph, (b) a Small-World Network and (c) a Scale-Free Network. 10

3.1 Illustration of conditions $I(i, j, k)$ and $O(i, j, k)$. 14

3.2 Impact of network topology over average consensus algorithm with single update in constant ω case. 19

3.3 Impact of network topology over average consensus algorithm with single update in balanced ω case. 19

3.4 Node degree distribution for RGG, SWN and SFN averaged over 500 graphes. 20

4.1 Communication in single update and multiple updates cases. 21

4.2 Illustration of conditions $E(i, j, k)$ and $D(i, j, k)$. 22

4.3 Performance comparison between average consensus algorithm with single update and multiples updates averaged over 100 simulations with $\lambda = 0.05$, $p_b = 0.2$, $T_b = 2$, $T_l = 10$. 26

4.4 Impact of initial value distribution over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $p_b = 0.2$, $T_b = 2$, $T_l = 10$. 28

4.5 Impact of initial value distribution over mean squared error for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $p_b = 0.2$, $T_b = 2$, $T_l = 10$. 28

4.6 Impact of listening probability over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $T_b = 2$, $T_l = 10$ with constant ω. 29

4.7 Impact of listening probability over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $T_b = 2$, $T_l = 10$ with constant ω. 30

4.8 Impact of listening probability and listening time interval over mean squared error for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $T_b = 1$, $\alpha = 0.5$ with constant ω. 31

4.9 Impact of λ and listening time interval over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $p_l = 0.8$, $T_b = 2$, $\alpha = 0.5$ with constant ω. 31

4.10 Impact of λ and broadcasting time interval over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $p_l = 0.8$, $T_l = 10$, $\alpha = 0.5$ with constant ω. 32

4.11 Impact of topology over average consensus algorithm with multiple update in the constant ω scenario. 33

5.1 Example of initial value distribution where nodes are in two distinct clusters with a frontier area in a network of 100 nodes. 36

5.2 Example of average consensus reached in two distinct clusters with a frontier area in a network of 100 nodes. 36

5.3 Example of path between x_i and the received value x_j. 37

5.4 Comparison of mean convergence time between clustering methods with constant ω in discrete split case 40

5.5 Comparison of mean number of broadcast between clustering methods with constant ω in discrete split case 41

5.6 Comparison of mean cluster accuracy between clustering methods with constant ω in discrete split case 41

5.7 Comparison of mean difference in number of cluster between clustering methods with constant ω in discrete split case 42

5.8 Comparison of local consensus algorithm mean convergence time with constant ω in discrete and continuous split case 42

5.9 Comparison of local consensus algorithm mean number of broadcast with constant ω in discrete and continuous split case 43

5.10 Comparison of local consensus algorithm mean clustering accuracy with constant ω in discrete and continuous split case 43

5.11 Comparison of local consensus algorithm mean difference in number of cluster with constant ω in discrete and continuous split case 44

List of Tables

3.1 Boolean conditions composing $C(i, j, k)$. 14

4.1 Boolean conditions composing $U(i, j, k)$. 22

List of Abbreviations

DQEB	**D**ynamic **Q**uery-tree **E**nergy **B**alancing
LEACH	**L**ow-**Energy** **A**daptative **C**lustering **H**ierarchy
Li-ion	**Li**thium **Ion** battery
RGG	**R**andom **G**eometric **G**raph
SFN	**S**cale-**F**ree **N**etwork
SWN	**S**mall **W**orld **N**etwork
WSN	**W**ireless **S**ensor **N**etwork
DBSCAN	**D**ensity-**B**ased **S**patial **C**lustering for **A**pplications with **N**oise
AC	**A**verage **C**onsensus

Glossary

- Clustering coefficient: Coefficient that indicate the trends of nodes in a graph to form cluster.
  - Global clustering coefficient: A way to compute the clustering coefficient of a graph by using the following formula

    $$C = \frac{3n_{triangle}}{n_{triplet}}$$

    where $n_{triangle}$ is the number of triplets of nodes fully connected and $n_{triplet}$ is the number of triplets in the graph.
  - Local clustering coefficient: A way to compute the clustering coefficient of a graph by using the following formula

    $$\forall v_i \in V,\, C_i = \frac{|\left\{e_{jk} : v_j,\, v_k \in N_i,\, e_{jk} \in E\right\}|}{k_i\left(k_i - 1\right)}$$

    $$C = \bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$$

    where N_i is the neigborhood of node i,k_i is the degree of node i, V is the ensemble of vertex and E is the ensemble of edge.
- Error: Event that occurs when a system or a component delivers a different output than the excepted output.
- Failure: Event that occurs when a system or a component can not perform its computations.
- Fault: Source of error and failure.
- Power law: Relation between two quantities x and y following the equation

  $$x = ay^b$$

  with a and b as parameters.
- Characteristic path length: Mean path length between two random nodes in a graph.
- Random geometric graph: Graph generated by putting node inside a metric space by following a given distribution and connecting them if the distance is below a certain threshold value.
- Scale-free network: Graph with a degree distribution that follows a power law at least asymptotically.

- Small-world network: Graph that show small-world properties, namely an high clustering coefficient and a low mean path length.

Introduction

A lot of effort has been done in the area of sensor network and especially in Average Consensus (AC) algorithm. In this work, we study some consensus algorithms for low-complexity nanosensor networks through asymmetric broadcasting. The consensus problem is a basic problem in distributed systems like sensor networks where all agents must agree on a single value.

This kind of sensors are limited to the minimal functions to provide a front-end to classical wireless sensor networks, allowing them to have a better control over their environment. In order to communicate, those nanosensors use asymmetric broadcasting transmission through gossiping which removes the need of rooting protocol or identification mechanism. This method of communication is inspired by the spread of gossip inside social networks.

They also need to use their resources effectively, especially energy, since their small size is hard limit to their capability. Because of those limitations, we focus on low-complexity sensors by allowing nodes to perform simple computations, broadcast their values or listen for broadcasts. Indeed, those functions are the minimal required to allow sensor network to reach consensus.

This kind of research is necessary for applications like neural dust presented in [14, 15] or robotic materials described in [9]. But, more generally, this research can be used in all application requiring ultra-dense sensor networks.

Indeed, the network is more and more solicited with the growing of the Internet of Things and the begining of the Internet of Materials. According to [7], we expect the number of connected devices to grow to more than 50 billion in 2020. The traffic resulting from this will provoke interference between devices and network congestions. Conventional technology now available seems to reach its limits, especially for high-density networks.

The goal of high-density networks is usually to perform statistical measurement from all nodes. According to this, we have developed consensus algorithm that are especially suited for this task. Consensus problem is a fundamental problem in distributed systems like sensor networks where all nodes must agree on a single value that is a function of all nodes values [10, 18–21].

In this work, we study two AC algorithms that follow this definition of a consensus algorithm. Then, we also introduce a clustering consensus algorithm where nodes solve the consensus problem inside their cluster. This variant can be useful for application where we want to detect a specific area like a part of a bridge undergoing extreme constraint.

The starting point of our work is the paper [12] from Peper. We present mathematical model of our two AC algorithms based on the framework given in [10]. We then analyse performances of our algorithms based on computer simulations run in Python 3 using Scipy and Numpy libraries.

The choice of Python for writing our code for computer simulations is based on several considerations. First, Python is a multi-platform programming language. So, we do not need to worry about the environment on which we will run it. Then, Python is a language often used in scientific works and some giant of Internet provide libraries for this language. For example, TensorFlow, the library for artificial intelligence from Google, is provided by default in Python. Another important aspect is its clear syntax making it easy to read even without Python-experience. Finally, Python enjoys a very active and useful community to help to solve issues encountered. There is many other reasons to choose Python for scientific works that are exposed in [11].

In this work, we will start by the state of the art in sensor network algorithms. In this part, we will present actual method to process data in sensor networks. Then, we will address some basic distributed algorithm to finally give an overview of randomized gossiping algorithms. In the second Chapter, we will define in-depth the limitation of our low-complexity sensors, expose the impact over their capabilities and explain why classical approach can not work with them. Chapter 3 will present the AC algorithm developed by Peper in [13]. Then, we will give a refined version of its mathematical model and present computer simulations to measure the network topology impact. We will introduce an improved version of this algorithm in Chapter 4 by defining a new update rule allowing multiple updates. We will then present its mathematical model based on the work of previous chapter and [10] and discuss of its performance based on numerical results. Chapter 5 will introduce a clustering convergence algorithm. We will first explain our motivations to develop this algorithm and then expose multiple working tracks. We will conclude this chapter by a discussion about the performance of this algorithm based on simulation results. Then, we will conclude this thesis with items for future works.

Chapter 1

Overview of average algorithms

In this chapter, we give an overview of data gathering and average algorithm in wireless sensor network. In the first section, we present a method with a central control mechanism. Then, we expose distributed algorithms with a more in-depth view of randomized gossiping and conclude with the requirements of presented algorithms. Based on this, we explain in Chapter 2 why they can not be used with constraint and limitation introduced in next chapter.

1.1 Centralized algorithms

The approach adopted by methods with a central control mechanism to compute statistical values is to gather data from each nodes and transmit it to a single node called sink node. When this node has collected all the data, it performs all computations to calculate statistical values like the average. The central problem in this method is the data gathering phase that consumes a lot of energy because some nodes need multiple hops to reach the sink node and then some nodes need to retransmit data from others nodes. To improve this method with a central control mechanism, two algorithm families has been developed to optimize data gathering.

The first family is tree-based family where a tree is built inside the network with the sink node at its root. There are different strategies used in this family but we will look more in detail for one of them, Dynamic Query-tree Energy Balancing (DQEB) presented in [22]. We choose to present DQEB because it is one of the principal tree-based algorithm and this algorithm already have the objective to minimize the cost of the data gathering phase. DQEB assumes a cluster structure. In each cluster, a node is selected to be the cluster head and aggregates data from other nodes inside its cluster. The role of the cluster head is to perform preliminary computations to send in a single transmission all the information from its cluster to minimize the number of transmission needed. Another assumption of DQEB is that a query-tree exists with the sink node at its root and all cluster heads as nodes. Then DQEB balance the energy consumption by changing the query-tree structure based on the remaining energy of cluster heads.

The second family is cluster-baser family like Low-Energy Adaptative Clustering Hierarchy (LEACH) presented in [5]. This family focus on how to build clusters to balance energy consumption. LEACH assumes that all nodes can reach the sink node and introduces randomized rotations of cluster heads. At each round, nodes

can randomly select themselves as cluster heads. After that, cluster heads transmit their status to other nodes that choose to which cluster they will belong based on minimal energy communication. The cluster is then built and the cluster head defines a transmission schedule followed by all nodes in its cluster to transmit data to it. Once the cluster head has all data from its cluster, it aggregates them and sends the result to the sink node.

To conclude this section, the goal in method with a central control mechanism is to balance the energy consumption during the gathering step. This is a very powerful method because once all data are gathered at the sink node, it can compute any statistical value but it also requires a lot of energy. The sink node also represents a single point of failure and needs to have storage and computation capabilities to trait all the data from the network. And those methods all require some local knowledge about the network topology.

1.2 Distributed algorithms

An alternative to centralized methods are distributed algorithms. One major problem in distributed computing is the consensus agreement problem. This problem is defined as a problem where all correct processes, which means all sensor nodes in our case, must agree on a single value. We can consider two main scenario based on the fact that all nodes share the same clock or that each node has its own clock. But, this only impacts the mathematical model of the algorithm and not how it works. This is why we will focus on how nodes aggregate values to reach consensus. This is call the fusion scheme.

A first naive fusion scheme is flooding. Xiao explain this fusion scheme in [18–21]. In this scheme, each node communicates only with its neighbours to exchange data and maintains a table with one value for each node in the network. A node can then eventually get the knowledge of all the network and perform the same computation as the sink node in algorithms with a central control mechanism. This fusion scheme is almost not studied because it requires large amount of memory and a great number of transmissions.

Another fusion scheme is pairwise algorithm. In this fusion scheme, nodes exchange their values inside a pair and update their values to their average. At each round, a node wakes up and contact one of its neighbour. This ensures that the sum of all values in the network does not change and preserves the average. If we assume that the network is a connected graph, the network converge to a consensus value because each node converges to a local consensus value and some nodes have a different neighbourhood than its neighbours. We assume two node i and j such as they are not neighbours but node k belongs to the neighbourhood of the two nodes. Then, node i and all its neighbours try to reach a consensus value x_i and node j and its neighbourhood does the same by converge to a consensus value x_j. Since node k belongs to each neighbourhood, it try to converge to x_i and x_j thus x_i and x_j must be equal considering that node k converges to a consensus value.

1.3 Overview of randomized gossiping

Randomized gossiping is a complex fusion scheme for distributed algorithm. First introduced by Tsitsiklis in [16], this stochastic algorithm gained a lot of interest in last decades with the growing interest for distributed system according to [10]. This algorithm is a gradient-like algorithm where nodes do not update their value to the optimal one based on their knowledge of the network but perform small improvement in the best direction given their knowledge. In all generality, this algorithm is given by (1.1) with N_i the neighbourhood of node i.

$$x_i(k+1) = x_i(k) + \epsilon \sum_{j \in N_i} \omega_{ij} \left[x_j(k) - x_i(k) \right] \tag{1.1}$$

An interpretation of (1.1) is that $x_i(k+1)$, the value of node i at step $(k+1)$, is equal to its previous value at step k plus a small improvement. In all generality, we can define this improvement by the sum of the difference between the value of node j and node i multiplied by the weight ω_{ij}. ϵ is a non-negative non-zero parameter for the step size.

The key point in randomized gossiping algorithm is how to define the value of ω_{ij}, the weight given to node j at node i or the weight given to the directed edge from node j to node i. In [18], Xiao and Boyd give the necessary and sufficient conditions over the matrix of all weights, W, to reach convergence and give an algorithm to compute optimal ω_{ij}. In this section, we will focus on three algorithms used to compute the matrix W. If you want more details, we recommend to read [18–21] from Xiao, [3] from Boyd and [2] from Avrachenkov.

The Best Constant Algorithm is an algorithm given in [18] to determine constant weight based on global knowledge of the network topology. With this algorithm, the weight is define by (1.2) with λ_i being the i^{th} largest eigenvalue of the Laplacian of the graph. Even if this solution is very easy to implement, the need of a global knowledge make it really sensitive to change inside the network and at least one node need to explore the whole network in order to compute weights.

$$\omega_{ij} = \frac{2}{\lambda_1 + \lambda_{n-1}} \tag{1.2}$$

Another algorithm requiring global knowledge of the network topology is the Max Degree Algorithm where each node needs to know d_{max}, the maximum node degree in the network. Based on this, all nodes will compute weights based on (1.3). Like the Best Constant Algorithm, this algorithm is easy to implement and does not have strong assumption on the node capabilities, like identification.

$$\omega_{ij} = \frac{1}{d_{max}} \tag{1.3}$$

Those two first algorithms to compute ω_{ij} are rather simple. Indeed, they are based on global knowledge of the network topology and they do not depend either of i or

of j. Indeed, their goal is to set a global weight ω that all nodes can use. But there are more complex solutions like the one presented after.

The Local Degree Algorithm, also know as Metropolis-Weight Algorithm, is a local algorithm to decide the weight w_{ij} introduced in [18]. It is an improvement of the Max Degree Algorithm where each sensor selects compute its weight based on the maximum between its own degree and its neighbours degree following (1.4). But this algorithm have also a strong requirement that node can identify their neighbours.

$$\omega_{ij} = \frac{1}{max(d_i, d_j)} \tag{1.4}$$

In conclusion, all of this algorithm try to define the best value for w_{ij} by looking at some measurement on the network topology. But, they also assume that $w_{ij} = w_{ji}$ and assign a weight to a given edge. Because of that, it appears that nodes need to be able to identify the source of a message to choose the right weight except for algorithm that use constant ones.

Chapter 2

Low-complexity nanosensors netkors

In this chapter, we give an overview of the limitations on low-complexity nanosensors. Then, we explain the impact of those limitation on network capabilities. After that, we conclude with reason we can not use algorithms presented in Chapter 1 in high-density networks with such sensors as nodes. In the Chapter 3, we present an algorithm that solve problems exposed in this chapter with algorithms presented in chapter 1.

Since Peper do not define clear characteristics for those nanosensor, we assume the following definition of a nanosensor. A nanosensor is a sensor with a submillimeter size and less capabilities than those of Class 0 Internet-connected devices according to the RFC 7228.

2.1 Limitations on low-complexity nanosensors

The limitations on low-complexity wireless sensor network is given by Peper in [13]. In this paper, he gives major limitations of this kind of network and present a maximum consensus algorithm. This is a trivial algorithm but most of his work is defining limitations for low-complexity nanosensors that are key elements in our research. We present here an overview of constraint in low-complexity nanosensor network.

2.1.1 Energy constraint

Like in conventional networks, most of the space available on each sensor is for energy supply. But we must also consider nanosensors being out of our range once they are spread. In other words, it is impossible due to the size of a nanosensor to perform maintenance operations on it like change or recharge its battery or connect it to an external power source once it is spread. Therefore, we only envisage energy storage and energy harvesting to provide energy to the nanosensor.

For energy storage, the best options are actually lithium-ion batteries and supercapacitors. Now, lithium-ion battery is the better option and an already proven technology. They are used in many industrial domain like smartphones, laptops and portable game consoles. Lithium-ion batteries are almost always chosen by manufacturers because they have an high energy density and a high number of life cycles. An alternative to Li-ion batteries could be supercapacitors. There is still a lot of research about this type of device but this technology is still in early development. For a more exhaustive review of energy storage system, we recommend to read [4].

On another hand, energy harvesting is also limited by the size of node. Most techniques have a minimal size needed to be implemented and the amount gathered is directly dependent to the size of node according to Peper in [13]. Therefore, we need to efficiently use this tiny energy budget and focus on the core function of our network. An interesting review about energy harvesting is made by Mateu in [8].

2.1.2 Computation power and memory constraint

Distributed computing always plays an important role in sensor networks but it becomes critical for nanosensor networks. Because of their size, nanosensor have a small area available for circuitry that limits their processing capacity. According to [13], we consider sensor with an available space for circuitry of the order of 1 mm^2 This constraint is also reinforced by the tiny energy budget because more complex processing needs more energy.

Memory is also strongly connected with area available for circuitry. Given that the available area can not be expanded, we need to make a compromise between computation power and memory. This research being theoretical, there is actually no device we can use as a standard. So, we decide to fix the memory capacity to the minimum needed by our algorithm. But, based on the RFC 7228, this should be less than 10 KiB.

2.1.3 Communication range constraint

The last impact of node small size is range limitation. This is due to the energy limitation and also to the physical size of the node. A problem with radio frequencies is that optimal frequency is inversely proportional to antenna size. Thus, nanosensor with extremely small antenna need high frequency. But higher frequency increases energy consumption and signal absorption from environment. Some solutions exist to solve this issue, like the proposition to use ultrasound in [15]. This solves partially the energy consumption problem but nanosensors can not have long range transmission.

2.1.4 Lack of identification

The last limitation on low-complexity nanosensor is the lack of identification. One requirements for identification is to be able to define an id for each node and ensure this id is unique. This uniqueness constraint is difficult to ensure because it requires

a knowledge of all id given in the network. Since the network topology can change, the only way to ensure id uniqueness is to use mechanism like those to attribute IP address. This represents an overhead in terms of number of communication but also in terms of computation complexity. As a result of what, Peper decides that nanosensors we study have no id.

2.2 Impact on network

In this section, we will present the major impact over the network of nanosensor limitations.

2.2.1 Communication routing

In network, we distinct two ways of sending message. The most used is unicast that consist in sending a message to a single target and the second way is broadcast where message is sent to all.

There is multiple mechanisms to allow unicast. The first one is routing table that is used on Internet for example. This allow unicast and multi-hop communication between nodes but this also require a lot of energy and memory. Given that those two resources are extremely limited in nanosensor, using routing table is very expensive.

Another way to perform unicast is random walk. Leibnitz presents a version of the random walk protocol in [6] and he gives an overview of this protocol. In random walk, one node has the message, initially the source node of the message. Then, at each round, the node currently having the message sends it to one of its neighbours randomly chosen. This process is repeated until the message reaches the target node. But, node need to be able to decide if they are the target or not to finish the random walk.

The random walk protocol shows why unicast can not be used in our network. Because of the lack of identification, it is impossible to define a single target into the network. Therefore, unicast is impossible.

If unicast is unworkable, nodes can only use broadcast to communicate. But, there is still one parameter to define, the maximum distance that can be covered by the broadcast. In other terms, this is the maximum number of hops that the broadcast message can do in the network.

A simple solution should be to set it to the infinity to spread the message through the whole network. Indeed, all nodes need values from all other nodes to compute the average. This is roughly what is performed by flooding algorithm presented in subsection 1.2.

Another solution is to set this maximum distance to one. This solution is inspired by the spread of gossip in social network. In social network, one person is at the source of a gossip and tells it to all its friends. Then, those friends can tell the gossip to their own friends with, eventually, some modifications. Transposed to sensor network,

one node performs a broadcast to send its value to all its neighbours. Then those neighbours can spread this value after they have aggregate it with their own value. This is exactly the way randomized gossiping algorithms presented in subsection 1.3 work.

2.2.2 Network topology

Network topology is an important characteristic of a network. On our work, Peper decides to focus on three graph families: Random Geometric Graph (RGG), Small-World Network (SWN) and Scale-Free Network (SFN).

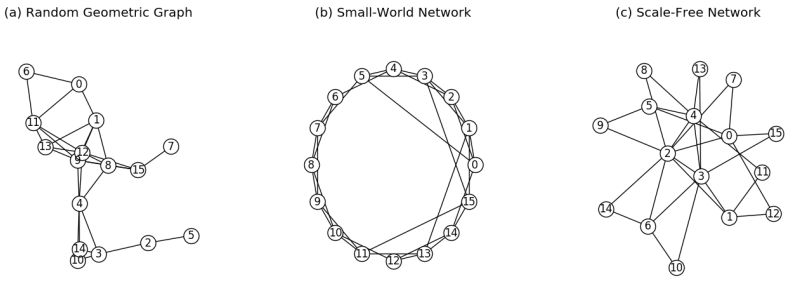


FIGURE 2.1: Illustration of (a) a Random Geometric Graph, (b) a Small-World Network and (c) a Scale-Free Network.

RGG is not strictly a graph family but it is an algorithm family to generate a random graph which is a graph family. In graph generated by RGG, nodes are randomly spread in a space with a distance metric. Two nodes are then connected if the distance between them is lesser than a given value like in Figure 2.1 (a).

SWN is a graph family introduced by Watts in [17]. This graph family is defined by graph having a small characteristic path length and a high clustering coefficient. This kind of graph is common in biological systems or in social networks. Examples of SWN given by Watts in his paper is the film actors graph where two actors are connected if they have been in a film together or the diseases spread graph. It appears that, in those graphes, any two random nodes are connected by a short path of neighbours like illustrated in Figure 2.1 (b). There is several algorithms to build SWN but we decide to use the Watts-Strogats algorithm presented in [17].

SFN is another graph family presented by Albert in [1]. SFN are characterized by a node degree distribution that follow, at least asymptotically, a power law. In other words, SFN are networks with a few nodes, called hubs, that present a very high node degrees like illustrated in Figure 2.1 (c). This is a graph family common in computer network like Internet and a generic way to generate a SFN is to use preferential attachment models like Barabàsi and Albert model given in [1].

A major problem with SWN and SFN is that there is no known algorithm to generate a spatial graph from those families and ensure it will respect the communication range limitation of nanosensors. Thus, we assume that network have only RGG topology for the rest of this work excepted if it is clearly said that we study a SWN

or SFN. We also assume that the graph is connected. This means that, for all pair of nodes in the graph, there is a path between them.

2.3 Effect on average algorithms

With the limitation given in section 2.1 and 2.2, it appears that most of the algorithm presented in Chapter 1 can not be used in low-complexity nanosensor networks.

First, algorithms presented with a central mechanism control prensented in 1.1 assume the existence of a routing tree. Given that there is no routing available in our network for the reasons exposed in subsection 2.2.1, this routing tree does not exist. As a direct consequence, nanosensor networks do not satisfy the requirements to use those algorithms.

The Flooding algorithm presented in subsection 1.2 seems to be a solution to this lack of routing mechanism like we say in subsection 2.2.1. But this solution presents a major drawback. Flooding algorithm requires a huge amount of memory to store values from all nodes in the network which is incompatible with the limited memory of nanosensors.

The Pairwise algorithm and randomized gossiping algorithm are then the best options for low-complexity nanosensor networks. Indeed, they do not need routing mechanism and requires only a limited memory. But those algorithms often require identification. In pairwise algorithm, identification is required to form a pair of nodes to exchange data. It is also required in randomized gossip algorithm to identify the source of a message and select the appropriate weight to update the node value. Those algorithms also assume symmetric communication, allowing node to communicate with their neighbours at each iteration.

In conclusion, the only algorithms presented in Chapter 1 that can be used in low-complexity nanosensor networks is randomized gossiping with constant weight. Indeed, it does not matter which node is the source in this version because all weights are identical.

Chapter 3

Average consensus algorithm with single update

In Chapter 2, we show that only randomized gossiping algorithms with constant weight are suitable for low-complexity nanosensor networks. Based on this algorithm, Peper presented an AC algorithm in [12]. In this chapter, we give an overview of this AS algorithm. Then, we show the mathematical model of this algorithm and present numerical results. Finally, an improvement of this algorithm is given in the next Chapter.

3.1 Algorithm presentation

The algorithm given by Peper in [12] belongs to the randomized gossiping family presented in subsection 1.3. In this family of algorithms there are two parameters that we can modify to impact the algorithm performance. The first one is the way we compute the matrix of weight W. It is on this parameter we focused in subsection 1.3 where we present different algorithms to compute this matrix. But, those algorithms assume a symmetric communication which is not true in our network.

A solution to this issue is then to divide the matrix W as a combination of two factors. The first one is ω_i, the weight given by node i to all broadcast it receive. The other one is a parameter a_{ij} that represent the probability for node i to receive a broadcast from node j. Then, we rename the iteration step-size ϵ as α, the learning rate of nodes.

To describe how this algorithm work, we first explain the comportment of a single node inside the network. The arrival of event in each node is considered as a Poisson process with a rate λ of marked events. Those events can be broadcasting events with a probability p_b or listening events with the complementarity probability $p_l = 1 - p_b$. Given that those operations need some time, we fix arbitrary time intervals for broadcasting and listening, respectively T_b and T_l. With fixed time intervals, it can occur that the node triggers a new event before it finishes the current event. In this case, we consider that the current event is lost.

Based on this node behaviour, we can define a boolean condition to represent if a node updates its value or not at an iteration k corresponding to the k^{th} event in the system. This condition is defined as follows with three terms defined in Table 3.1

and illustrated by Figure 3.1:

$$C(i,j,k) \equiv L(i,k) \wedge I(i,j,k) \wedge \forall r \in N_i \setminus \{j\} \neg O(i,r,k)$$

With the given definition of $C(i,j,k)$, node i make an update at iteration k only if it enters in listening state at iteration k and is not interrupted by another event of its own, one of its neighbours sends a broadcast message that fall inside this listening interval and no other nodes sends message to node i that overlaps its listening interval. In other word, node i at iteration k makes an update only if it enters in listening state at this iteration and receives a single broadcast message that fully falls inside its listening interval.

Boolean condition	Definition
$L(i,k)$	Node i enters in listening state at iteration k and is not interrupted by another event of its own.
$I(i,j,k)$	A broadcast from node j falls inside a listening interval of node i at iteration k.
$O(i,j,k)$	One or more broadcast from node j overlap a listening interval of node i at iteration k.

TABLE 3.1: Boolean conditions composing $C(i,j,k)$.

Based on this interpretation of $C(i,j,k)$, a node can only receive one broadcast message and make only one update each time it enters in listening state. Because of this, we call this algorithm the AC algorithm with single update.

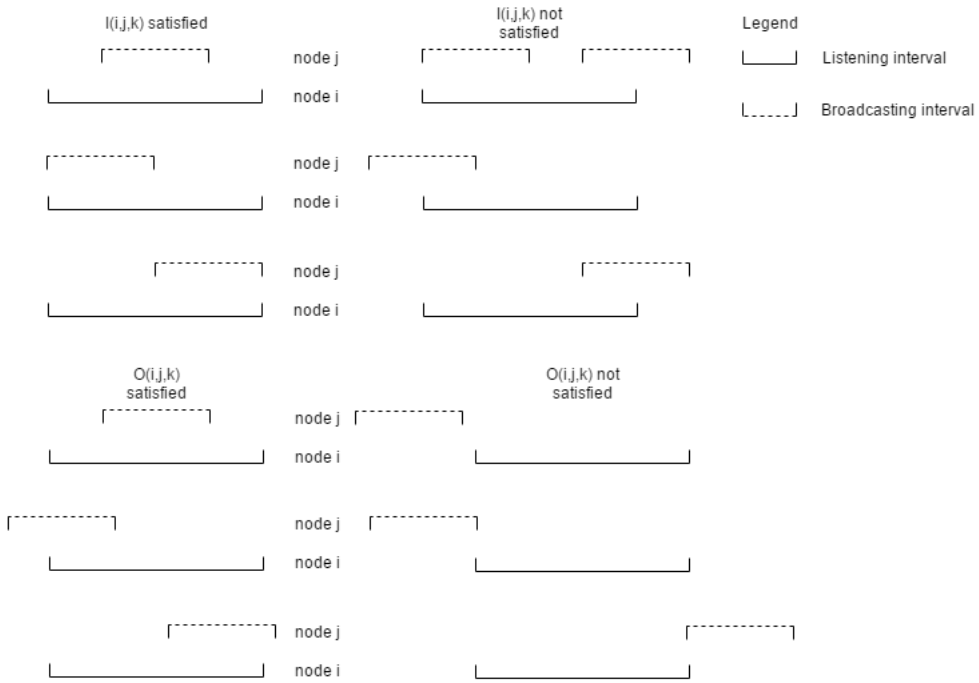


FIGURE 3.1: Illustration of conditions $I(i,j,k)$ and $O(i,j,k)$.

3.2 Mathematical model

This mathematical model has been given by Peper in [12] but we review it for the sake of the next section. We also add some minor modifications to reflect that iteration k correspond to the k^{th} event in the network. Thus, we do not need to keep trace of all node individual clocks unlike Peper in its model.

The node behaviour described in the previous section can be mathematically described using equation (1.1) like all randomized gossiping algorithms. Peper renames the step size ϵ in this equation by the learning rate α. This parameter comes from biological systems and is common in systems inspired by nature. We can find this parameter in neural network for example. An intuition of the meaning of this parameters is how quick the node will forget the past to adapt new values. With this modification, we can describe the evolution of x_i, the value of node i as follow:

$$x_i(k) = \begin{cases} x_i(k-1) + \frac{\alpha}{\omega_i}\left[x_j(k-1) - x_i(k-1)\right] & \text{If } C(i,j,k) \\ x_i(k-1) & \text{Else} \end{cases} \tag{3.1}$$

Node i is unaware of the sending node according to the definition of $C(i,j,k)$ and this equation. In order to analyse the general behaviour of the system, we average all the possible scenarios. Mathematically, we define $m_i(k) = E[x_i(k)]$ the expected value of $x_i(k)$.

$$\begin{aligned} m_i(k) = &\left\{1 - \sum_{j \in N_i} Pr[C(i,j,k)]\right\} \frac{\alpha}{\omega_i} m_i(k-1) \\ &+ \sum_{j \in N_i} \left\{ Pr[C(i,j,k)] \left[m_i(k-1) + \frac{\alpha}{\omega_i}\left(m_j(k-1) - m_i(k-1)\right)\right]\right\} \end{aligned} \tag{3.2}$$

In order to give a precise definition of this equation, we need to define the probability of condition $C(i,j,k)$. Based on its definition, we determine it as:

$$Pr[C(i,j,k)] = Pr[L(i,k)]\ Pr[I(i,j,k)] \prod_{r \in N_i \setminus \{j\}} \left\{1 - Pr[O(i,r,k)]\right\}$$

Then, we must calculate the probability of boolean condition $L(i,k)$, $I(i,j,k)$ and $O(i,j,k)$ to provide a defined equation for $C(i,j,k)$. $L(i,k)$ is satisfied if node i is the source of event at iteration k, it chooses to enter in listening state with a probability p_l and no other event of its own occurs during a time interval T_l. Since those three probabilities are independent and events at node i follow a Poisson distribution with rate λ, we can define the probability of $L(i,k)$ as:

$$Pr[L(i,k)] = \frac{1}{n}\, p_l\, e^{-\lambda T_l}$$

with n the number of nodes in the network.

Condition $I(i,j,k)$ is satisfied if a broadcast of node j falls inside a listening interval of node i based on its definition. In other words, that means that node j should start a broadcasting event inside the first part with length $T_l - T_b$ of listening interval. This ensures that there is enough time inside the listening interval to receive the complete broadcast and avoid overlapping. Then, no other broadcast events of node j should occur in the remaining part with length T_b of the listening interval. Those two events are independent so we obtain for $I(i,j,k)$:

$$\begin{aligned} Pr[I(i,j,k)] &= p_b\lambda(T_l - T_b)e^{-p_b\lambda(T_l-T_b)}e^{-p_b\lambda T_b} \\ &= p_b\lambda(T_l - T_b)e^{-p_b\lambda T_l} \end{aligned}$$

because broadcasts of node j are Poisson distributed with rate $p_b\lambda$.

Finally, $O(i,j,k)$ is satisfied if one or more broadcastings of node j overlaps a listening interval. In other words, it means that one or more broadcast events of node j must start inside the listening interval or in the interval of length T_b just before. Indeed, if a broadcast starts inside this interval, it finishes inside the listening interval. The probability of $O(i,j,k)$ is then equal to one minus its complementary probability which is the probability there is no broadcast event of node j inside an interval $(T_b + T_l)$. Using the distribution rate $p_b\lambda$ of broadcast event at node j, we determine:

$$Pr[O(i,j,k)] = 1 - e^{-p_b\lambda(T_b+T_l)}$$

Based of those values for $Pr[L(i,k)]$, $Pr[I(i,j,k)]$ and $Pr[O(i,j,k)]$, we can provide a value for $Pr[C(i,j,k)]$. Using its value given previously and replacing all probabilities inside it, we obtain the following probability:

$$\begin{aligned} Pr[C(i,j,k)] &= Pr[L(i,k)]\ Pr[I(i,j,k)] \prod_{r\in N_i\setminus\{j\}} \{1 - Pr[(O(i,r,k)]\} \\ &= \frac{1}{n}\ p_l\ e^{-\lambda T_l} p_b\lambda(T_l - T_b)e^{-p_b\lambda T_l} \prod_{r\in N_i\setminus\{j\}} \left\{1 - \left[1 - e^{-p_b\lambda(T_b+T_l)}\right]\right\} \\ &= \frac{1}{n}\ p_l\ p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l} \prod_{r\in N_i\setminus\{j\}} e^{-p_b\lambda(T_b+T_l)} \\ &= \frac{1}{n}\ p_l\ p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l} e^{-p_b\lambda(T_b+T_l)(|N_i|-1)} \end{aligned}$$

By reintroducing this value in equation (3.2), we can then write:

$$\begin{aligned} m_i(k) = &\left[1 - \frac{|N_i|}{n}\frac{\alpha}{\omega_i}\ p_l\ p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l} e^{-p_b\lambda(T_b+T_l)(|N_i|-1)}\right] m_i(k-1) \\ &+ \frac{1}{n}\frac{\alpha}{\omega_i}\ p_l\ p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l} e^{-p_b\lambda(T_b+T_l)(|N_i|-1)} \sum_{j\in N_i} m_j(k-1) \end{aligned}$$

Then, we can rewrite the system behaviour in matrix form by defining $m(k) = \left[m_1(k), \ldots, m_n(k)\right]^T$. We then obtain

$$m(k) = S\, m(k-1) \tag{3.3}$$

where matrix S as a square matrix of order n with its elements

$$s_{ii} = 1 - \frac{|N_i|}{n}\frac{\alpha}{\omega_i}\, p_l\, p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l}e^{-p_b\lambda(T_b+T_l)(|N_i|-1)}$$

in the diagonal,

$$s_{ij} = \frac{1}{n}\frac{\alpha}{\omega_i}\, p_l\, p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l}e^{-p_b\lambda(T_b+T_l)(|N_i|-1)}$$

outside the diagonal if node j belong to the neighbourhood of node i and

$$s_{ij} = 0$$

otherwise.

With this description of the system behaviour in matrix form, we can use the theoritical framework provided by Ofti-Saber in [10] to analyse the performance of our algorithm. The construction of the matrix S ensure that S is a stochastic matrix, which means that all its row-sums or column-sums equal 1. Indeed, we have that

$$\forall i, \sum_j s_{ij} = 1$$

Then, based on theorem 2 in [10], we have that the system converge to a fixed point of equation (3.3) and an average consensus is asymptotically reached if S is a double-stochastic matrix, which means that all its row-sum and column-sum are 1.

In order to preserve the sum of all node on average in the update process, we need to balance the w_i to ensure that

$$\forall j, \sum_i s_{ij} = 1$$

As explained by Peper in [12], this requires that the inner product of the **1**-vector and column i of matrix S equals 1. Mathematically, we express this as:

$$\mathbf{1}^T s_{\bullet i} = 1 - \frac{1}{n}\alpha\, p_l\, p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l}\sum_{j\in N_i}\left[\frac{e^{-p_b\lambda(T_b+T_l)(|N_i|-1)}}{\omega_i} - \frac{e^{-p_b\lambda(T_b+T_l)(|N_j|-1)}}{\omega_j}\right] = 1$$

According to his work, we need to find values for each pairs of nodes i and j such as the following is true.

$$\frac{\omega_j}{\omega_i} = \frac{e^{-p_b\lambda(T_b+T_l)(|N_j|-1)}}{e^{-p_b\lambda(T_b+T_l)(|N_i|-1)}}$$

Since node i do not have any information about node j, we can satisfy this equality by setting $e^{-p_b\lambda(T_b+T_l)(|N_i|-1)}/\omega_i$ constant. We then define ω_i has

$$\omega_i = c\, e^{-p_b\lambda(T_l+T_b)(|N_i|-1)} \tag{3.4}$$

with c a constant independent from node i. By applying this equation in previous equality to ω_i and ω_j, we can assert that the equality holds. If we apply theorem 2 in [10] with system in equation (3.3), we obtain that system converges to average consensus if the following condition is true.

$$0 < \alpha < \min_{i=1\ldots n} \frac{\omega_i\, n}{|N_i|\, p_l\, p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l} e^{-p_b\lambda(T_b+T_l)(|N_i|-1)}}$$

Then, substituting the ω_i by the balanced ones given by equation (3.4), we can rewrite this condition as:

$$0 < \alpha < \min_{i=1\ldots n} \frac{c\, n}{|N_i|\, p_l\, p_b\lambda(T_l - T_b)e^{-(1+p_b)\lambda T_l}} \tag{3.5}$$

By replacing the value of w_i given by equation (3.4) in (3.2). We obtain that

$$s_{ij} = \frac{\alpha p_l\, p_b\lambda(T_l - T_b)\, e^{-(1+p_b)\lambda T_l}}{c\, n}$$

for $i \neq j$ and node j being in the neighbourhood of node i. Equation (3.5) then implies

$$s_{ij} < \frac{1}{\max\limits_{r=1\ldots n} |N_r|}$$

3.3 Numerical results

In this section, we present computer simulations of our AC algorithm with single update. Fot this, we well consider the two following scenarios to define ω_i:

1. ω_i set to the constant value 1. In this scenario, ω_i are not balanced and, according to [10], the algorithm converges but not necessary to the average of all node values. In the rest of this section, we will refer to this scenario as "constant ω".
2. ω_i set to the value given by (3.4) with constant c define as $\max\limits_{r=1\ldots n} e^{p_b\lambda(T_l+T_b)(|N_i|-1)}$. With this value of c, the algorithm converges to the average of all node values according to [10]. This scenario will be referred as "balanced ω" in this section.

Peper present already some computer simulations of this average consensus algorithm with single update in [12]. Since he exposes basic aspects of this algorithm in it, we focus on the impact of network topology. The result presented in 3.2 are averaged over 50 simulations with following parameters:

- $T_b = 2$
- $T_l = 10$

- $\lambda = 0.05$
- $p_b = 0.2$

and error bars represent the 95% confidence interval.

In order to evaluate the impact of network topologies, we generate random network for three type of network. We use RGG with 100 nodes and a communication radius of 0.2 like in [12] as reference type of network. We also use Watts-Strogatz model given in [17] with 100 nodes, a mean node degree of 20 and a rewriting probability of 0.01 to generate SWN. And SFN are generated using the Barabàsi-Albert model presented in [1] with 100 nodes, 5 initial nodes and 5 edges built at each iteration.

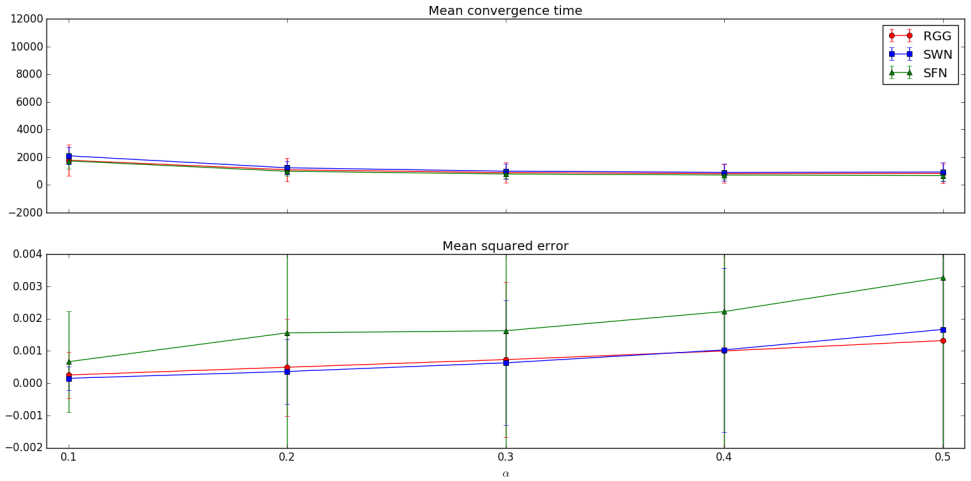


FIGURE 3.2: Impact of network topology over average consensus algorithm with single update in constant ω case.

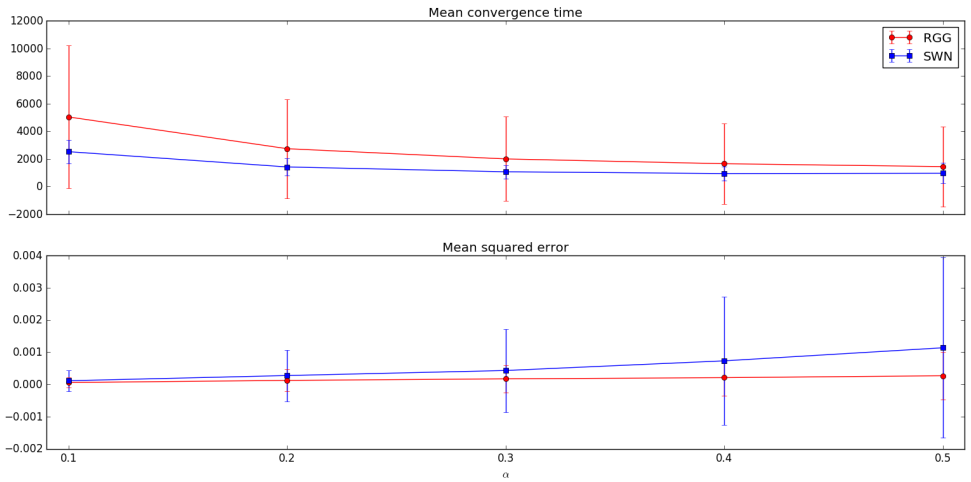


FIGURE 3.3: Impact of network topology over average consensus algorithm with single update in balanced ω case.

According to result from figures 3.2 and 3.3, we observe that network topology have an impact on the mean convergence time. To explain the gap observed between the

constant ω and the balanced ω scenario, we can use the formula for ω_i given by (3.4) and replacing the value given to c in our scenario. Then, we can express ω_i as follow:

$$\omega_i = e^{p_b \lambda (T_l + T_b)(\max\limits_{r=1\dots n}(|N_r|) - |N_i|)}$$

Based on this, we can conclude that the convergence speed is inversely proportional to the maximum difference between node degrees. Since this difference is smaller in Small-World Network than in other network according to Figure 3.4, SWN are less impacted by the choice between constant or balanced ω.

We also assume, from the definition $\omega_i = e^{p_b \lambda (T_l + T_b)(\max\limits_{r=1\dots n}(|N_r|) - |N_i|)}$, that a lot of node in SFN will have a very low update rate. If we refer to Figure 3.4, we can expect a difference of the order of 50 for more than half of the nodes between the maximum node degree and their own node degree. This mean that, for half of the node in SFN, $\omega_i \approx e^2 400$ with our parameters setting. Since the update rate is equal to α/ω_i, we can see that this value is of the order of 10^{-1043} which is almost 0. Based on Equation (3.1), we can finally conclude that those nodes will converge extremely slowly. This is why we decided to not run simulations with SFN and balanced ω.

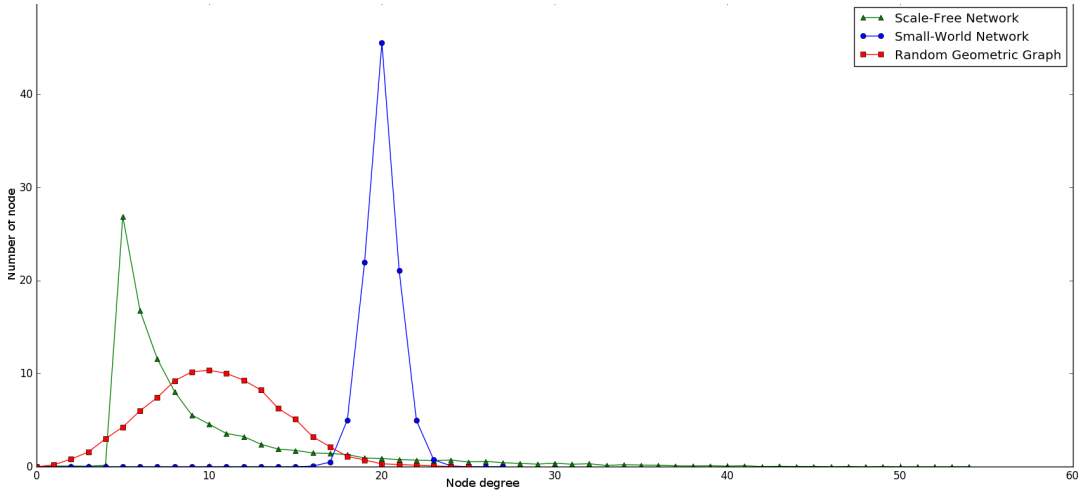


FIGURE 3.4: Node degree distribution for RGG, SWN and SFN averaged over 500 graphes.

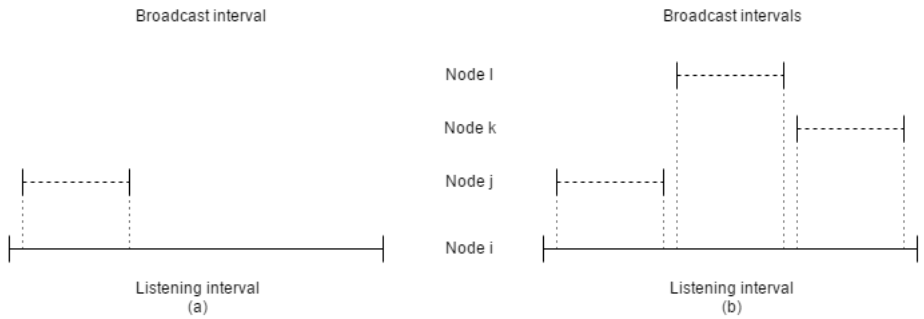
Chapter 4

Average consensus algorithm with multiple updates

In this Chapter, we present an improvement of the algorithm presented in Chapter 3. First, we define what is multiple updates and how we expect this to improve our AC algorithm. Then, we give the mathematical model of this improved algorithm and expose numerical results. Based on this algorithm, a clustering consensus algorithm is introduced in Chapter 5

4.1 Algorithm presentation

In the algorithm presented in Chapter 3, we allow a node to make a single update inside a listening interval. But it is possible to have multiple broadcasts that fall inside a listening interval without overlapping. We should then allow a node to perform an update for each broadcast it receive during its listening interval if those broadcast does not collide. To illustrate this difference, Figure 4.1 shows a correct transmission with single update (a) and multiple updates (b).



(a) Communication in single update case. (b) Communication in multiple updates case.

FIGURE 4.1: Communication in single update and multiple updates cases.

In order to improve our algorithm and model this change, we replace the condition $C(i, j, k)$ given in the previous Chapter by another condition $U(i, j, k)$. This condition is also composed of three boolean conditions given in 4.1 and illustrated by

Figure 4.2. The condition $U(i, j, k)$ is then defined as follow:

$$U(i, j, k) \equiv B(i, k) \wedge E(i, j, k) \wedge \forall r \in N_j \neg D(j, r, k)$$

Based on this definition of $U(i, j, k)$, there is a clear change in our point of view. In the mean consensus algorithm with single update, we focus on the node that receives the message while we concentrate on the node sending it in this new version. Those two points of view are complementary and lead to the same final mathematical model. But the broadcaster point of view is more suited to develop the mathematical model of our AC algorithm with multiple updates.

Boolean condition	Definition
$B(i, k)$	Node i enters in broadcasting state at iteration k and is not interrupted by another event of its own.
$E(i, j, k)$	A listening interval from node j encapsulates a broadcast of node i at iteration k.
$D(i, j, k)$	One or more broadcast from node j overlaps a broadcast interval of node i at iteration k.

TABLE 4.1: Boolean conditions composing $U(i, j, k)$.

With this new update condition, we expect a drop of the collision numbers to improve the convergence speed of our algorithm. Indeed, the condition $U(i, j, k)$ allows a single node to perform multiple updates during a single listening interval. Thus, we will no longer consider there is a collision in a situation like the one illustrated in Figure 4.1 (b).

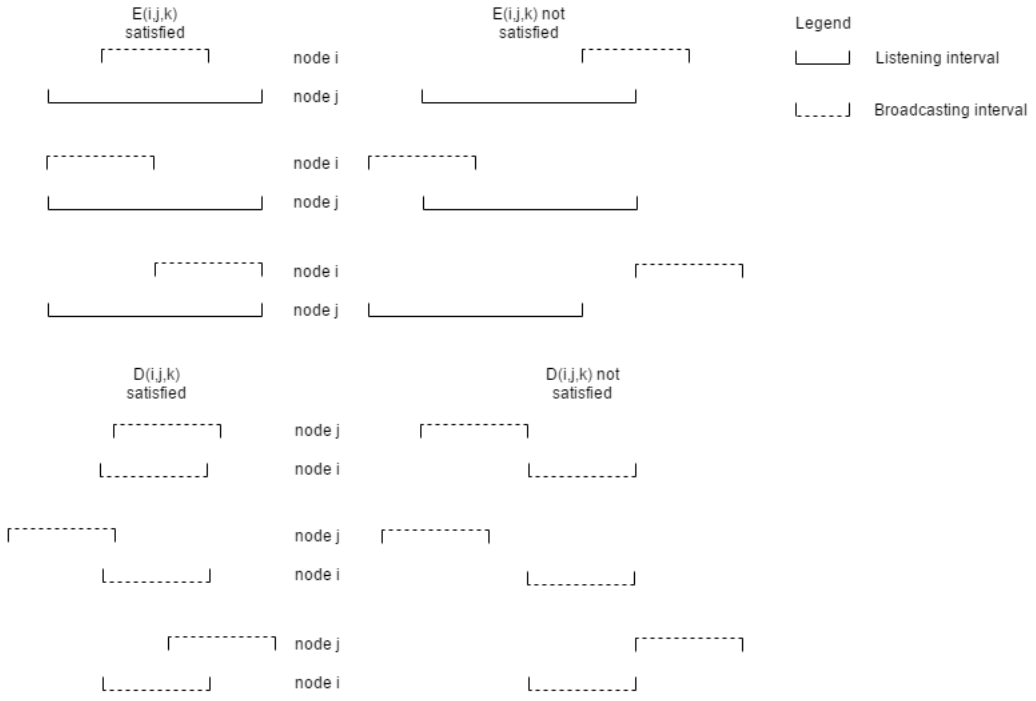


FIGURE 4.2: Illustration of conditions $E(i, j, k)$ and $D(i, j, k)$.

4.2 Mathematical model

The mathematical model of this algorithm uses the same mathematical framework given in [10] than the single update algorithm. But, because of our different point of view for this model, a little trick is needed. Then, instead of applying the condition $U(i,j,k)$, we use the condition $U(j,i,k)$ in the following equation:

$$x_i(k) = \begin{cases} x_i(k-1) + \frac{\alpha}{\omega_i}\left[x_j(k-1) - x_i(k-1)\right] & \text{If } U(j,i,k) \\ x_i(k-1) & \text{Else} \end{cases}$$

Indeed, this trick corresponds to a rename of variable and allows us to keep node i and j in their roles of respectively receiving node and sending node. Then, in a similar way to Chapter 3, we average all the possible scenarios for $x_i(k)$ by defining $m_i(k) = E[x_i(k)]$, the expected value of $x_i(k)$.

$$\begin{aligned} m_i(k) = &\left\{1 - \sum_{j \in N_i} Pr[U(j,i,k)]\right\} \frac{\alpha}{\omega_i} m_i(k-1) \\ &+ \sum_{j \in N_i} \left\{ Pr[U(j,i,k)] \left[m_i(k-1) + \frac{\alpha}{\omega_i}\left(m_j(k-1) - m_i(k-1)\right)\right]\right\} \end{aligned} \tag{4.1}$$

At this point, the equation representing the two algorithms are really close. Indeed, the only change in the system behaviour is the update rule but the rest remains the same. The first step is then to define $Pr[U(i,j,k)]$ which is equal to

$$Pr[U(i,j,k)] = Pr[B(i,k)]\ Pr[E(i,j,k)] \prod_{r \in N_j \setminus \{i\}} \left\{1 - Pr[D(i,r,k)]\right\}$$

To give a proper definition of $Pr[U(i,j,k)]$, we need to derive the probabilities of $B(i,k)$, $E(i,j,k)$ and $D(i,j,k)$. Given its definition, $B(i,k)$ is true when node i is the source of event at iteration k, it enters in broadcasting state with a probability p_b and no other event of its own occurs during a time interval T_b. Those events being statistically independent we can give the given formula for $Pr[B(i,k)]$:

$$Pr[B(i,k)] = \frac{1}{n} p_b\, e^{-\lambda T_b}$$

with n the number of nodes in the network.

Condition $E(i,j,k)$ is met when the broadcast of node i is encapsulate inside a listening interval of node j. In other words, it means that node j must start a listening event during an interval of length $T_l - T_b$ before the beginning of broadcasting. This ensures that node j will receive the whole broadcast if there is no other event of its own during a time interval of length T_L. Then, we can give the following equation for this probability:

$$\begin{aligned} Pr[E(i,j,k)] &= p_l\lambda(T_l - T_b)e^{-p_l\lambda(T_l-T_b)}e^{-\lambda T_l} \\ &= p_l\lambda(T_l - T_b)e^{-\lambda\left[(1+p_l)T_l - p_l\ T_b\right]} \end{aligned}$$

Finally, condition $D(i,j,k)$ is met if at least one broadcast of node j overlaps a broadcast of node i at iteration k. This means that node j start a broadcasting event that is not interrupted during the broadcasting interval of node j or in an interval of T_b before it. To simplify this probability, we use its complementary probability which is the probability of no broadcasting event in an interval of length $2T_b$. We obtain then the following value:

$$Pr[D(i,j,k)] = 1 - e^{-2p_b\lambda T_b}$$

Replacing those values in the formula of $Pr[U(i,j,k)]$ we obtain the following result:

$$\begin{aligned} Pr[U(i,j,k)] &= Pr[B(i,k)]\ Pr[E(i,j,k)] \prod_{r\in N_j\setminus\{i\}} \left\{1 - Pr[D(i,r,k)]\right\} \\ &= \frac{1}{n}p_b\ e^{-\lambda T_b}\ p_l\lambda(T_l - T_b)e^{-\lambda\left[(1+p_l)T_l - p_l\ T_b\right]} \prod_{r\in N_j\setminus\{i\}} \left[1 - (1 - e^{-2p_b\lambda T_b})\right] \\ &= \frac{1}{n}p_b\ p_l\lambda(T_l - T_b)e^{-\lambda\left[(1+p_l)T_l - p_lT_b + T_b\right]} \prod_{r\in N_j\setminus\{i\}} e^{-2p_b\lambda T_b} \\ &= \frac{1}{n}p_b\ p_l\lambda(T_l - T_b)e^{-\lambda\left[(1+p_l)T_l + (1-p_l)T_b\right]}e^{-2p_b\lambda T_b(|N_j|-1)} \end{aligned}$$

Based on this, we can use this formula for $Pr[U(i,j,k)]$ and proceed to a rename of variable to define $Pr[U(j,i,k)]$. Then, we can introduce this value in (4.1) to obtain:

$$\begin{aligned} m_i(k) = &\left\{1 - \frac{|N_i|}{n}\frac{\alpha}{\omega_i}\ p_b\ p_l\lambda(T_l - T_b)e^{-\lambda\left[(1+p_l)T_l + (1-p_l)T_b\right]}e^{-2p_b\lambda T_b(|N_i|-1)}\right\} m_i(k-1) \\ &+ \frac{1}{n}\frac{\alpha}{\omega_i}\ p_b\ p_l\lambda(T_l - T_b)e^{-\lambda\left[(1+p_l)T_l + (1-p_l)T_b\right]}e^{-2p_b\lambda T_b(|N_i|-1)} \sum_{j\in N_i} m_j(k-1) \end{aligned}$$

Like in previous chapter, we can then define $m(k) = \left[m_1(k), \ldots, m_n(k)\right]^T$ to write the system behaviour in matrix form. This is write as:

$$m(k) = S\ m(k-1) \tag{4.2}$$

where matrix S as a square matrix of order n with its elements

$$s_{ii} = 1 - \frac{|N_i|}{n}\frac{\alpha}{\omega_i}\ p_b\ p_l\lambda(T_l - T_b)e^{-\lambda\left[(1+p_l)T_l + (1-p_l)T_b\right]}e^{-2p_b\lambda T_b(|N_i|-1)}$$

in the diagonal;

$$s_{ij} = \frac{1}{n}\frac{\alpha}{\omega_i}\, p_b\, p_l \lambda (T_l - T_b) e^{-\lambda\left[(1+p_l)T_l + (1-p_l)T_b\right]} e^{-2p_b\lambda T_b(|N_i|-1)}$$

for elements outside the diagonal if node j is a neighbour of node i and

$$s_{ij} = 0$$

otherwise.

By construction, this matrix is stochastic because it verifies

$$\forall i, \sum_j s_{ij} = 1$$

Therefore, our AC algorithm with multiple updates converges to a fixed point of equation (4.2). If we want to preserve the sum of all nodes on average in the update process, theorem 2 in [10] add as a necessary condition that matrix S is double-stochastic. Like for the average consensus algorithm with single update, this requires that the inner product of the $\mathbf{1}$-vector and column i of matrix S equals 1. Mathematically, this means that the following equality holds:

$$\begin{aligned} \mathbf{1}^T s_{\bullet i} &= 1 - \frac{1}{n}\frac{\alpha}{\omega_i}\, p_b\, p_l \lambda (T_l - T_b) e^{-\lambda\left[(1+p_l)T_l + (1-p_l)T_b\right]} \\ &\quad \cdot \sum_{j \in N_i} \left(\frac{e^{-2p_b\lambda T_b(|N_i|-1)}}{\omega_i} - \frac{e^{-2p_b\lambda T_b(|N_j|-1)}}{\omega_j}\right) \\ &= 1 \end{aligned} \tag{4.3}$$

To ensure the previous equality holds, a sufficient condition is to find values of ω_i and ω_j such that the following is true.

$$\frac{\omega_j}{\omega_i} = \frac{e^{-2p_b\lambda T_b(|N_j|-1)}}{e^{-2p_b\lambda T_b(|N_i|-1)}}$$

Then, we can define ω_i to make $e^{-2p_b\lambda T_b(|N_i|-1)}/\omega_i$ constant for the same reasons as in Chapter 3. Then, we define ω_i as

$$\omega_i = c\, e^{-2p_b\lambda T_b(|N_j|-1)} \tag{4.4}$$

with c a constant that do not depends of node i. According to theorem 2 in [10], we have that (4.1) with this values of ω_i converges to the average consensus if the following condition is met.

$$0 < \alpha < \min_{i=1\ldots n} \frac{\omega_i\, n}{|N_i|\, p_b\, p_l \lambda (T_l - T_b) e^{-\lambda((1+p_l)T_l + (1-p_l)T_b)} e^{-2p_b\lambda T_b(|N_i|-1)}}$$

We can replace ω_i by its value given by (4.4) to rewrite this condition as

$$0 < \alpha < \min_{i=1\ldots n} \frac{c\, n}{|N_i|\, p_b\, p_l \lambda (T_l - T_b) e^{-\lambda((1+p_l)T_l + (1-p_l)T_b)}} \tag{4.5}$$

4.3 Numerical results

In this section, we present computer simulations to compare performances of our AC algorithms with single update and with multiple updates. For those evaluations, we consider two different scenarios.

1. ω_i set to the constant value 1. In this scenario, ω_i are not balanced and, according to [10], the algorithm converges but not necessary to the average of all node values. In the rest of this section, we will refer to this scenario as "constant ω".

2. ω_i set to the value given in Chapter 3. We also use the same value of c than in the previous chapter, c being defined as $\underset{r=1...n}{max}\ e^{p_b\lambda(T_l+T_b)(|N_i|-1)}$. With this value of c, the algorithm converges to the average of all node values according to [10] for the average consensus algorithm with single update. This scenario will be referred as "balanced ω" in this section.

We do not use the balanced ω defined in this Chapter by equation (4.4) because this mathematical model has been developed after the computer simulations prove that it improve the algorithm performance. Furthermore, our goal is to compare performances of AC algorithms with single update and with multiples updates all other things being equal.

Results in this section are obtained by averaging simulations and error bars represent the 95% confidence interval. If we do not mention the graph topology used, we assume an underlying Random Geometric Graph (RGG) of 100 nodes and a communication radius of 0.2, nodes being uniform randomly distributed in the unit square.

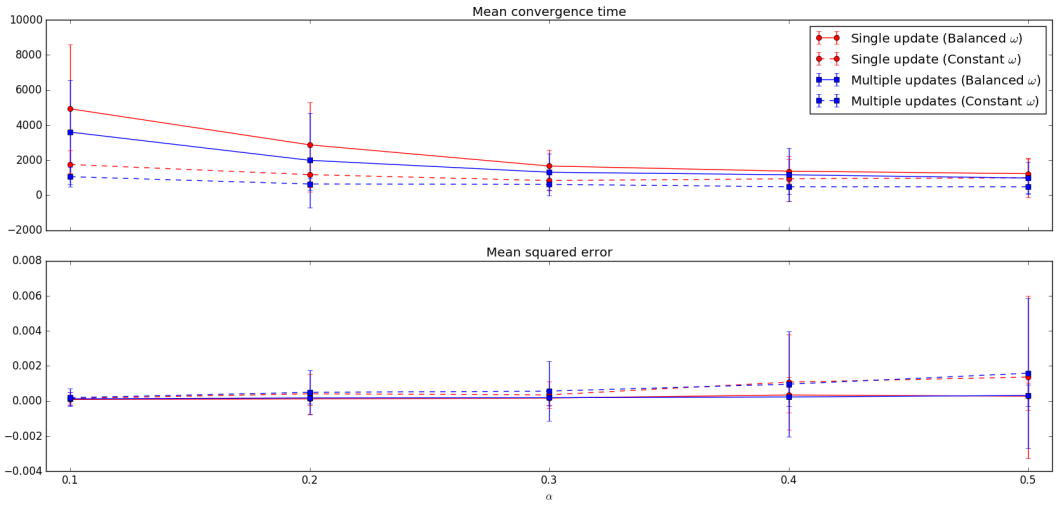


FIGURE 4.3: Performance comparison between average consensus algorithm with single update and multiples updates averaged over 100 simulations with $\lambda = 0.05$, $p_b = 0.2$, $T_b = 2$, $T_l = 10$.

According to our expectations, Figure 4.3 shows a clear improvement of convergence speed in both case while the error remains the same with the AC algorithm with multiple updates. Based on this, we can deduce that value of ω_i given in Chapter

3 verifies the necessary condition to reach average consensus for multiple updates. Since the optimal ω_i for this algorithm is difficult to evaluate, this allow us to use balanced ω from Chapter 3 to analyse the performance of the AC algorithm with multiple updates.

Once this improvement is confirmed, we decide to study the impact of the initial value distribution over the performance of our algorithm. In previous simulations, we always use an uniform distribution in interval $[0, 1]$ as the initial distribution. In order to measure impact of this initial distribution, we have considered the following scenarios.

1. Uniform case: Initial node values are set given an uniform distribution over the interval $[0, 1]$.
2. Gaussian case: Initial node values are set given a Gaussian distribution of mean 0.5 and standard deviation of 1.
3. Split case: Initial node values are set given the node position. Nodes with a x-coordinate below 0.5 are set given a Gaussian distribution of mean 0 and standard deviation of 0.025, otherwise they are set given a Gaussian distribution of mean 1 and standard deviation of 0.025.
4. Inverse-square law case: Initial node values are set given the distance between node and a randomly point of the unit square selected as the source following the inverse-square law. Then, all result are rescale to set the maximum initial value to 1.

Uniform and Gaussian cases are basic scenarios we chose because of their simplicity. Indeed, uniform distribution and Gaussian distribution are common in statistics. Split and inverse-square law cases are at the opposite because they correspond to more realistic scenarios. In the split case scenario, we envisage the network being over two distinct areas. For example, we can imagine to have nodes that measure temperature in a wall and having different values because of some underlying structure like one part of the wall being isolated and the other one not. In this scenario, the Gaussian distribution represent the noise on the measurement. The inverse-square law case corresponds to a general scenario where nanosensors measure some physical quantity from a source. The inverse-square law is a physical law stating that a physical quantity or intensity is inversely proportional to the square of the distance from the source of this quantity or intensity. An example of this law is the Newtow's law of universal gravitation that is defined as

$$\vec{F} = G\frac{m_1 m_2}{d^2}$$

The results in figure 4.4 shows that AC algorithm with multiple updates is sensitive to the initial distribution. A first explication to this observation comes from the behaviour of our algorithm. At each iteration, nodes update their values in order to be closer to the average. By doing this, they reduce the standard deviation at each iteration and we consider the consensus reached once this standard deviation is below a certain threshold. Given that Gaussian case has a standard deviation of 1, it needs more iterations than uniform case, for which the standard deviation is $1/\sqrt{12}$ according to the theory, to reduce it below the threshold value.

Another explication for this impact is the spatial distribution of initial values. In split case, nodes are divided into two clearly defined cluster. Inside those cluster, the consensus is almost reached but in both clusters they are far from average. In this situation, consensus is reached thanks to border nodes of each cluster that broadcast to nodes in the other cluster. However, this communication between nodes is counterbalanced by broadcast to border nodes from nodes inside their own cluster. Because of this, the convergence speed decreases drastically at the border between node clusters. Since the consensus is reached only thanks to convergence on this specific area, this decreases the convergence speed on the whole network.

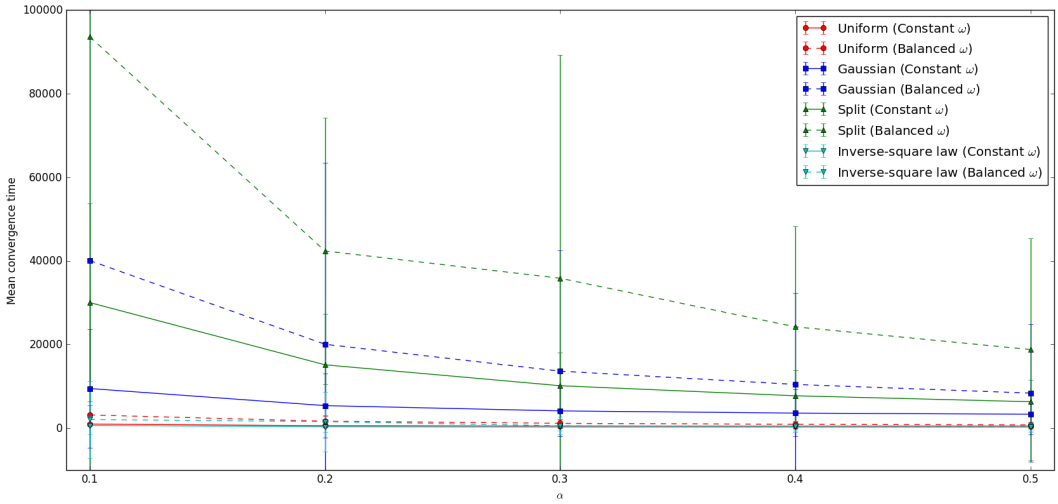


FIGURE 4.4: Impact of initial value distribution over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $p_b = 0.2$, $T_b = 2$, $T_l = 10$.

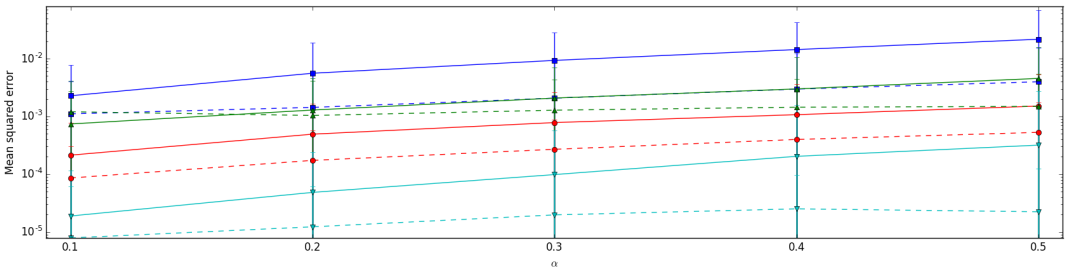


FIGURE 4.5: Impact of initial value distribution over mean squared error for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $p_b = 0.2$, $T_b = 2$, $T_l = 10$.

Thus, we can deduce from figure 4.4 that initial value distribution have an impact on convergence speed. The clustering of value inside the network is a factor that impacts convergence speed. Or, in other words, we can assume that the convergence speed is proportional to the spatial distribution homogeneity. We can also observe that the convergence speed is inversely proportional to the standard deviation of the initial value distribution. But, we can not conclude about their impact on the algorithm accuracy based on Figure 4.5. Indeed, the error variations remain not significant.

Another parameter to consider is the listening probability p_l and the broadcasting probability p_b. Since those two probabilities are complement complementary, we will focus on p_l and we define $p_b = 1 - p_l$. Figure 4.6 show the numerical results by step of 0.1 for α and 0.1 for p_l. Another view of those result are given in figure 4.7 for some selected values of p_l.

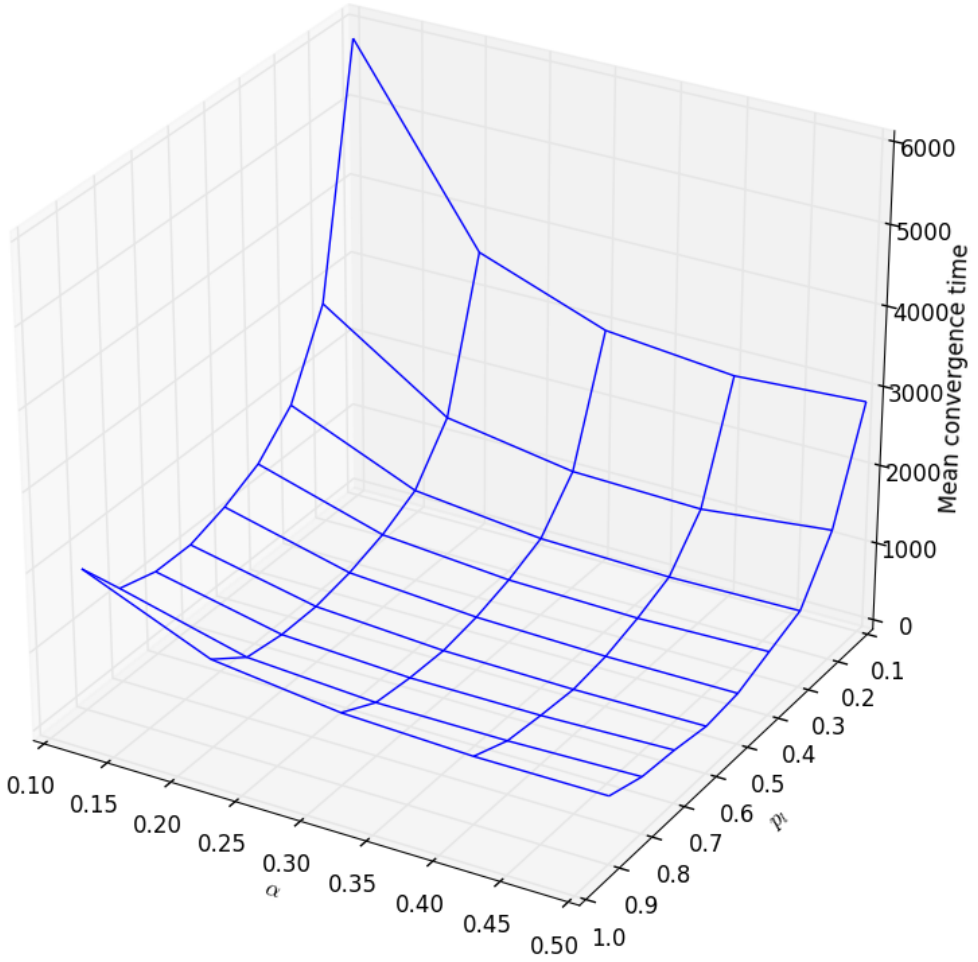


FIGURE 4.6: Impact of listening probability over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $T_b = 2$, $T_l = 10$ with constant ω.

According to figures 4.6 and 4.7, we can see that p_l as an impact on the convergence speed. It appears that there is an optimal value of $p_l = 0.6$. This suggest that p_l optimal value is linked to some other parameters of our algorithm. This assumption is based over the fact that there is a value of p_b where nodes make enough broadcast to take advantage of the multiple updates rule but do not broadcast too often to avoid network saturation and multiple message collisions.

Based on figure 4.7, we could also consider the possibility of an optimal value of α. However, we only have one data point from $p_l = 0.2$ that indicates this possibility. Since this data point has also a large confidence interval, it appears to be uncertain

to assume something from it. Then, further investigations will be needed to confirm or infirm this hypothesis. But we have decided to consider only value of α less than or equal to 0.5 within the framework of this thesis.

If we assume a link between optimal value of p_l and another parameters of our algorithms, most evident is T_l because there is a direct link between those two parameters. Indeed, they have an impact on $Pr[E(i, j, k)]$ and then an impact over the update probability.

In order to verify this hypothesis, we perform simulations where we vary T_l between 2 to 5 by step of 0.5 and p_l between 0.4 to 0.8 by step of 0.2. The result obtained are then presented in Figure 4.8. They show that, contrary to our assumption, there is no link between T_l and p_l or, at least, no direct link. Indeed, the three curves on the graph have exactly the same shape and appears to follow the same trend.

Based on results from figures 4.7 and 4.8, we can consider two options. The first one, p_l is an independent parameters that have no correlation with other parameters of our algorithm. In the other case, p_l is a parameter linked to λ or T_b. Indeed, they are the only parameters which with we do not have tested the listening probability. The first option seems rather unlikely because we naturally assume a link between the broadcasting interval T_b and the broadcasting probability p_b. Given that $p_b = 1 - p_l$, we conclude that there is a link, at least indirect, between T_b and p_l. Based on our intuition, λ could be the parameter that links all other but further investigations are required.

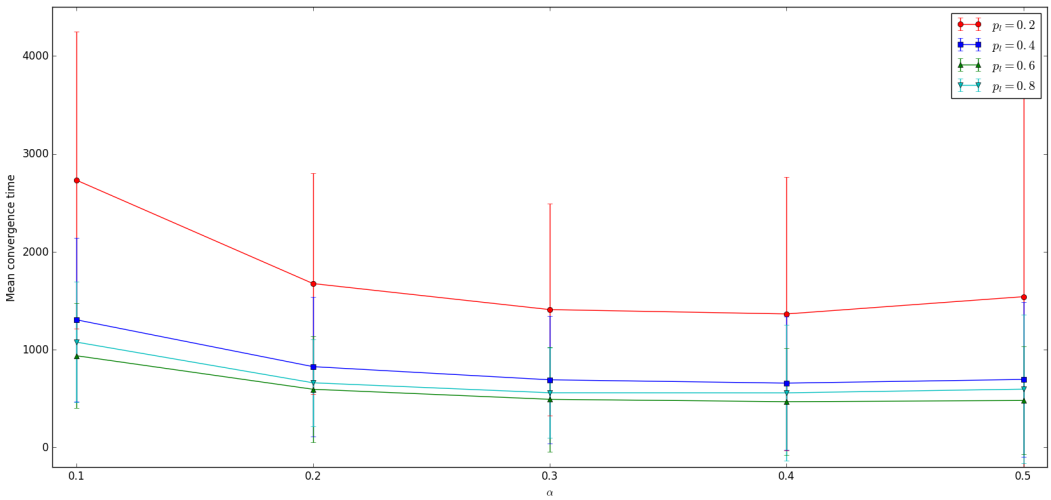


FIGURE 4.7: Impact of listening probability over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $T_b = 2$, $T_l = 10$ with constant ω.

Three other parameters that have intuitively a correlation between them are λ, T_b and T_l. It is difficult to produce clear figures to expose the link between three parameters. Indeed, figures like Figure 4.8 can show results for two parameters but it is difficult to visualize more parameters and keep something clear. So, we decide to define $T_b = 2$ only consider two of the three parameters, λ and T_l, in a first time.

Figure 4.9 shows there is a clear link between λ and T_l. Even if we can not define

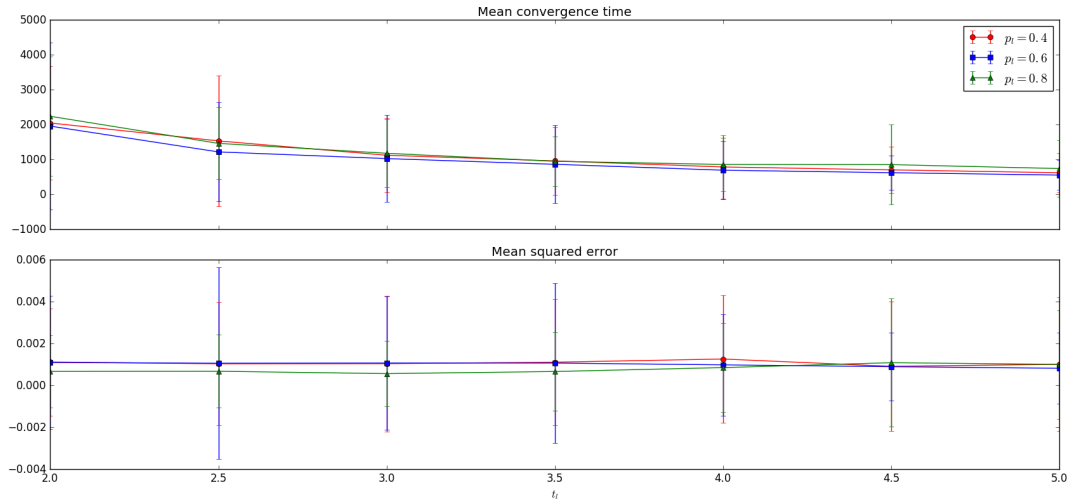


FIGURE 4.8: Impact of listening probability and listening time interval over mean squared error for average consensus algorithm with multiple updates averaged over 50 simulations with $\lambda = 0.05$, $T_b = 1$, $\alpha = 0.5$ with constant ω.

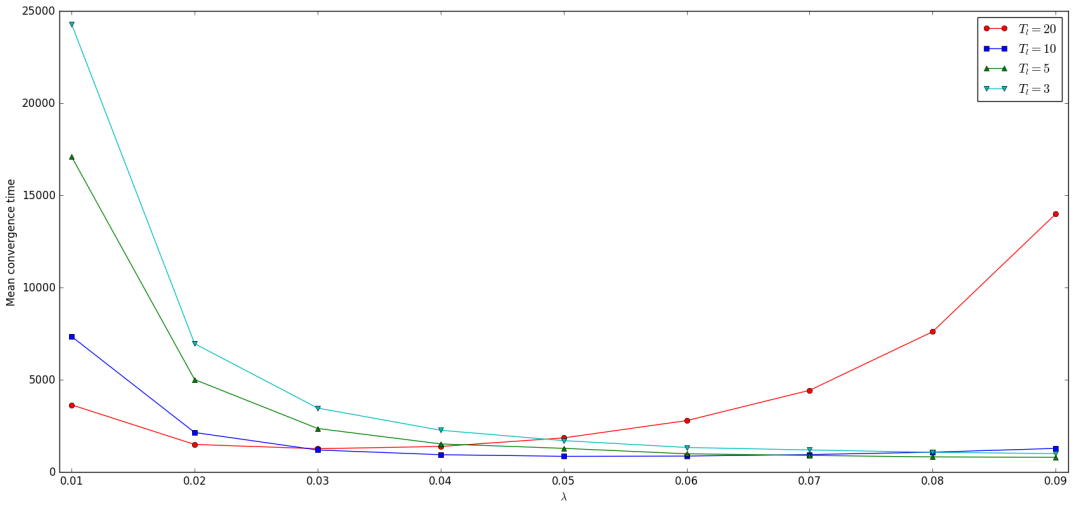


FIGURE 4.9: Impact of λ and listening time interval over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $p_l = 0.8$, $T_b = 2$, $\alpha = 0.5$ with constant ω.

a mathematical relation between those two parameters, we deduce that the optimal value of λ is inversely proportional to the value of T_l. This appears to be a normal conclusion since we need that no event occurs during a time interval T_l to have a listening event. Thus, λ must be small enough to have this happen. On the other hand, the rate of events is very important for the convergence speed. Then, λ must be great enough to not slow down the convergence. Given those two considerations, we can conclude there is an optimal value that correspond to the best trade-off between those two situations.

To confirm those results, we can perform a similar experience by setting $T_l = 10$ and

varying λ and T_b. For this experiment, we vary λ from 0.01 to 0.12 by step of 0.01 and T_b from 1 to 4 by step of 1. Based on the results given by Figure 4.10, we can deduce that λ and T_b are dependent. Intuitively, we can expect that λ must be small enough to avoid cancelled broadcasting and collisions. But, λ must be great enough to have a sufficient number of broadcast to reach consensus or the algorithm will be too slow. Like for T_l, there is thus a trade-off between those two aspects.

Finally, we could conclude from those results that there is an optimal value of T_b and T_l that depend of λ. We suspect also the existence of an optimal value of p_l also depending ofλ but we can not prove it with our current computer simulations. There is still a lot of work to do in the comprehension of links between the algorithm parameters and especially a lot of mathematical work to find formulas defining optimal parameters setting given a value of λ.

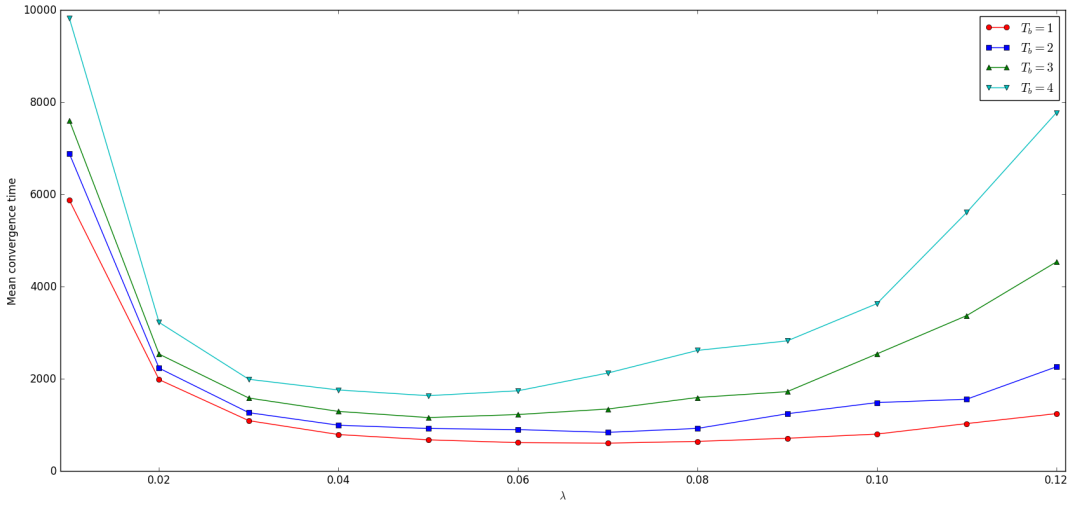


FIGURE 4.10: Impact of λ and broadcasting time interval over mean convergence time for average consensus algorithm with multiple updates averaged over 50 simulations with $p_l = 0.8$, $T_l = 10$, $\alpha = 0.5$ with constant ω.

A last aspect we want to analyse is the topology impact. Indeed, we show in chapter 3 that the average consensus algorithm with single update is sensitive to the network topology. We then reproduce the experience from previous chapter in the constant ω scenario. For this, we use the same RGG that in previous simulations in this Chapter as a reference network. Then, we also consider SWN of 100 nodes and a mean degree of 20 generated by the Watts-Strogatz model with a rewriting probability of 0.01. We also study SFN generated using the Barabàsi-Albert model with 100 nodes, starting from 5 nodes and building 5 edges at each iteration.

We have then performed this experiment with two parameter setting. The first parameter setting is the same that in Chapter 3 which is considered by Peper as an optimal parameter setting for the AC algorithm with single update. As a reminder, we set $\lambda = 0.05$, $T_l = 10$, $T_b = 2$ and $p_l = 0.8$ in this test scenario. For the second parameter setting, we keep the same parameters except p_l that we change to its optimal value of 0.6 according to results show in Figure 4.6.

The results of those simulations are shown in figure 4.11. Based on this, we can see

that our AC algorithm with multiple updates is sensitive to network topology like the single update version with the first parameter setting. But we can also observe that, with a proper parameter setting, the AC algorithm with multiple updates becomes insensitive to the network topology. Then, we could assume that parameter setting do not have an impact only over the convergence speed and accuracy but also over the resilience of our algorithm to the network topology.

This unexpected characteristic is extremely important because, in our simulation, we assume a specific and fixed network topology. But it is a strong precondition. Indeed, we explained in Chapter 2 that nodes have a limited amount of energy and then a limited lifetime. Thus, the network topology can change during the process. If we have a parameter setting where nodes only need to adjust their weight ω to adapt to a specific network topology, we can then expect to make an algorithm that dynamically updates node weights to be resilient to changes in the network topology.

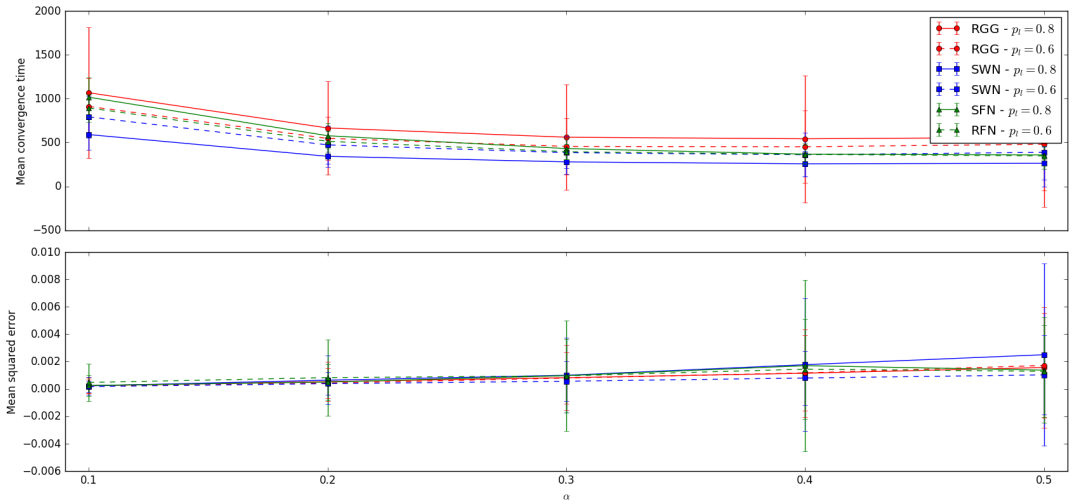


FIGURE 4.11: Impact of topology over average consensus algorithm with multiple update in the constant ω scenario.

Chapter 5

Clustering consensus algorithm

Based on the algorithm given in chapter 4. We present our motivations to develop a clustering consensus algorithm. Then, we explain how the algorithm works and finally we show numerical results.

5.1 Motivations

In previous chapters, we proved that our algorithm for average consensus works. But computing the average is a simple task and there is a lot of situation where we do not want to compute it on the whole network. An example of that is in robotic materials, one possible application of low-complexity nanosensor network, presented by McEvoy in [9]. To illustrate this, we can imagine a scenario of a bridge with robotic materials. The goal of the network is to detect an area under stress to adapt the structure of the bridge in order to balance the charge on it and avoid breaking.

Basically, there is a lot of scenario where you want to detect a specific area. This can go from the source of fire or chemical contamination to detect area of activity in the brain. But it can also be something very simple like nodes in different rooms and, possibly, different conditions. Typically, we wand to find average inside a cluster in a case where values are distributed like in the split case presented in chapter 4.

5.2 Algorithm presentation

The goal of this algorithm is to allow node to reach consensus inside their cluster. For this algorithm, we assume a scenario where initial values are split between two clusters as illustrated in figure 5.1. This scenario is of course an extreme case but we can consider this like a hot room in the red area and a cold one in the blue. Between them, in yellow, we have a wall that lets the heat pass through it but slowing down the energy transfer between those two rooms. Then, if the red room is warmed and the blue one cooled, this kind of equilibrium can happen.

Based on this, the goal of our algorithm is to reach consensus inside different clusters like in figure 5.2. In order to reach this objective, we decide to develop a clustering consensus algorithm where nodes decide to update their values or not when they

receive a broadcast. In the rest of this section, we expose some possibilities we have considered and explain their advantages and disadvantages.

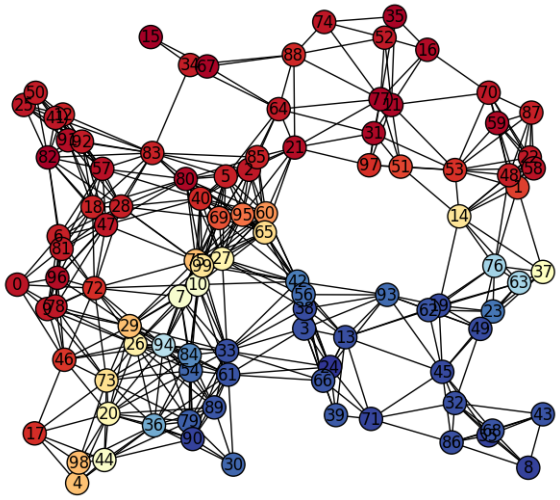


FIGURE 5.1: Example of initial value distribution where nodes are in two distinct clusters with a frontier area in a network of 100 nodes.

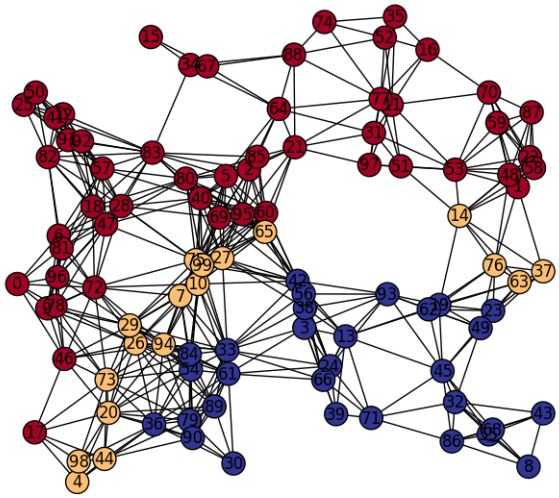


FIGURE 5.2: Example of average consensus reached in two distinct clusters with a frontier area in a network of 100 nodes.

A first idea is a naive algorithm. In this scenario, node i have a confidence interval of $[x_i - \tau; x_i + \tau]$ where x_i is the current value of node i and τ is a threshold value. If the value received falls inside the confidence interval, then node makes an update, else it drops the message received.

The advantage of this algorithm is its simple implementation that requires a limited amount of resources. But, this also have the disadvantage that we assume a value distribution where a majority of values will be inside the confidence interval. Thus, we need to have some insight over the initial value distribution to correctly set the parameters τ. Indeed, a too small value will cause the algorithm to make a lot of small clusters that does not exist. On the other hand, a too big value of τ will allow the algorithm to reach AC on the whole network.

An improved version of the previous solution can be implemented by adding the notion of path. In this algorithm, node i store in memory the value of the last l values it receives through broadcast. Then, when it receive a new broadcast, node i tries to build a path from its own value to the received value through stored values by successive steps of maximal size τ. Figure 5.3 illustrate a path built from value x_i and the received value x_j. Each color line correspond to the maximum possible step at each hop and the color arrow represent the selected hop. Since there is a path between x_i and x_j in this case, node i will update its value. Once the node has determined if it must make an update or not, it will throw the oldest value in its memory and replace it by the received value x_j.

The advantage of this path method is that we reduce the impact of the initial value distribution over the algorithm. Indeed, this mechanism of path building can be seen as a mechanism to dynamically adapt the confidence interval based on the node neighbourhood. In this sense, it is an improvement of the naive method but it also requires more memory and processing to build the path. Another disadvantage of the path method is the lack of way to determine a correct value of τ. Indeed, we do not need to have prior knowledge on the value distribution but we need to set a value of τ great enough to avoid creation of artificial clusters but small enough to avoid clusters merging. Unfortunately, our best option here is the trial and error method.

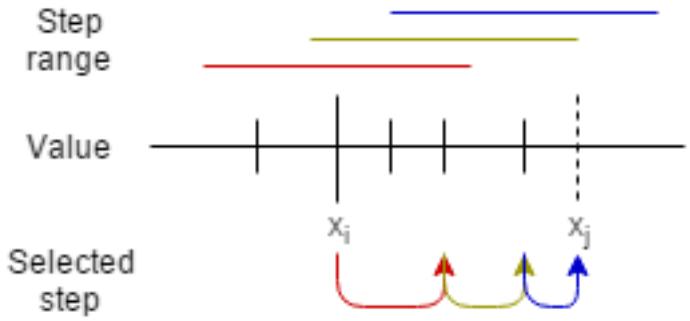


FIGURE 5.3: Example of path between x_i and the received value x_j.

A last basic solution we investigated is to vary the learning rate α at each node. The idea was to make node i fit to its physical reality by making the learning rate inversely proportional to the standard deviation inside the neighbourhood of node i. This translate in the network the reality of a physical separation between clusters. In simulation presented in the next section, we have computed it by having the node storing the last l received values and compute the standard deviation on it. In practice, we could theoretically use distributed computing to extract this statistical information like we do for the average. But this choice enables us a faster implementation in order to give a first analysis of this solution. Furthermore, we choose to alter this solution to avoid having the standard deviation slowing down convergence on the whole network. We then decide to apply a scaling parameter σ to the update weight ω_i when standard deviation in the neighbourhood of node i is above a threshold value τ.

This solution has the advantage that it requires nothing for us to determine. But this solution has also its drawback because there is still communication between clusters.

So, the system will eventually reach a global consensus if it has enough time and no corrective mechanism.

Finally, we proposed a last solution that from another point of view. Instead of trying to determine if the received value is from a node in their cluster or if their are a border nodes, nodes try to determine if they have reached a local consensus or not. To do that, a node computes the standard deviation of its neighbourhood and, if this value is below a certain threshold τ, the node flag itself as having reached consensus. When flagged, nodes do not broadcast or listen anymore. By this process, node in a cluster freeze their value to preserve the cluster average consensus while node at the border between clusters try to reach a consensus expected to be between the value of clusters. This approach from another angle is very interesting because we already use standard deviation of all node values in order to determine if consensus is reached. So, we apply it locally instead of globally to let nodes individually assess whether they have reach consensus or not in order to preserve natural clustering inside our network.

Like the standard deviation method, this local consensus method does not need any knowledge about the value distribution. Indeed, we already determine the threshold value τ which is when we consider consensus is reached. This method also solves the issue of communication between clusters of the previous standard deviation method. But, this algorithm lost a lot in accuracy as a counterpart because nodes near the border of the cluster tend to reach consensus with border nodes to form a clear separation between clusters. Another problem with this algorithm is that some nodes, especially those with a low node degree, never reach a local consensus because their neighbours reach it before them and then they never receive new values to update the standard deviation.

5.3 Numerical results

Based on the different solutions exposed in the previous section, we provide here a first rough analysis of performance. For this, we consider only two toy cases which are describe bellow.

- Discrete split case: This case is the same that the split case presented in chapter 4 for the initial value distribution analyse. In this scenario, nodes with an x-coordinate below 0.5 are set to 0, otherwise they are set to 1. We add to this Gaussian noise following a Gaussian distribution of mean 0 and standard deviation of 0.025.
- Continuous split case: This scenario is a slight modification of the split case. In it, we define a point as the source and node receive given their distance to the source following the given formula

$$f(x) = 1 - \left[\frac{\gamma(k, \frac{x\sigma}{\theta})}{\Gamma(k)}\right]$$

where x is the Euclidean distance between node and source, σ is a scaling parameter set to 20 and with $k = 11$ and $\theta = 20$. This function correspond to

the survival function of the gamma distribution. We choose this function with those parameters because it is characterized by being a continuous function and having two part with clearly distinct values. This curve shape allow us to represent a spit distribution with an area of transition between high and low value instead of a distribution with a rough separation between them.

To analyse our four algorithm variants, we chose four performance metrics. The two firsts are the mean convergence time and mean number or broadcast. Then, we compared clustering obtained with our algorithm and clustering obtained from Density-Based Spatial Clustering for Applications with Noise (DBSCAN). To do that, we run DBSCAN on the network with its initial values and then we run it a second time after convergence is reached. Then we compare the two clusterings based on the difference of the number of cluster and the cluster accuracy, or in other words the proportion of nodes that are in the same cluster in the two clusterings.

To determine if consensus is reached, we run DBSCAN after each iteration over the network after removing the nodes with a frozen value in the Local Consensus algorithm. Then, we compute the standard deviation in each cluster and we consider consensus reached if standard deviation is under a given threshold value in each cluster.

All the computer simulations performed to provide the result exposed in the rest of this section used the following parameter setting that we consider it respect the constraint given in Chapter 2

- $T_l = 2$
- $T_b = 10$
- $\lambda = 0.05$
- $p_l = 0.8$
- $\tau = 0.05$
- $\sigma = 0.1$ (Standard Deviation)
- Buffer size = 10 (Path, Standard Deviation, Local Consensus)
- Number of simulation for each data point = 50

A first observation from Figure 5.4 is that most efficient algorithms are the naive algorithm and the path algorithm. An explication of this result is that those two algorithm are especially suited for situation with a clear separation between clusters whereas standard deviation and local consensus algorithms try to deal with a non-existent transition area. Indeed, border node in the standard deviation algorithm converge slowly due to the scaling parameters and the influence of nodes from the other cluster. Local consensus algorithm do not have this issue but, instead, it build a transition area that form a new cluster where a consensus to a new value must be reached. The algorithm is thus slowed down be the formation of these unexpected transition areas.

We can also deduce from Figures 5.4 and 5.5 that number of broadcast required to reach consensus is directly proportional to the mean convergence time. This relation is perfectly logic because there is a direct relation between those two metrics. Indeed, the broadcast events follow a Poisson process of rate $p_b\lambda$. So, the more time we need to reach consensus, the more more broadcast we perform. But we can highlight that local consensus algorithm allow to spare a lot of broadcasting and then energy. The reason of this is that, in this algorithm, some nodes freeze themselves after assessing they have reached consensus. Thus, those nodes do not broadcast and, if there is less nodes broadcasting, the number of broadcast decrease.

Finally, we can remark in those two Figures a peak for the standard deviation algorithm around $\alpha = 0.2$. Actually, we do not know if it is the result of a too small number of simulations with some extreme case in it, something related to the algorithm itself or related to our way to determine that consensus is reached.

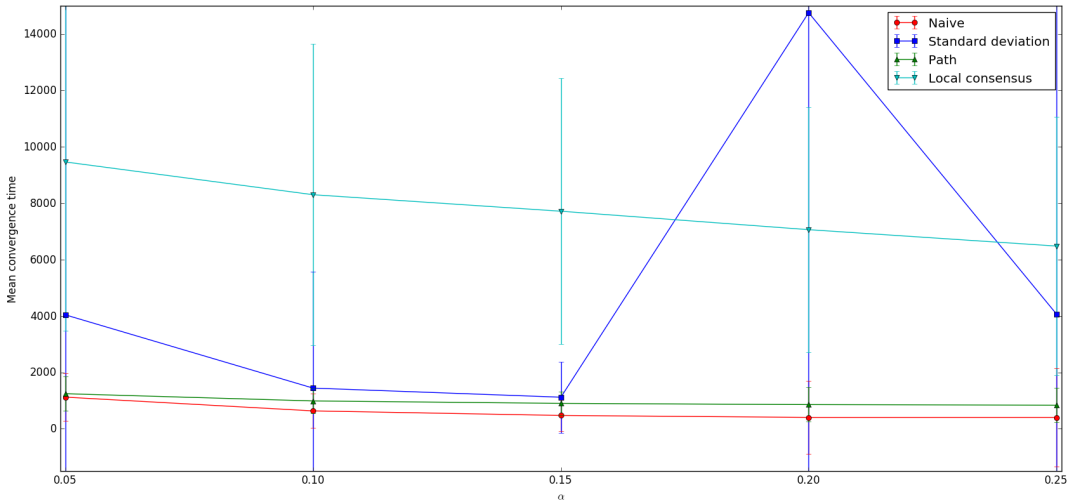


FIGURE 5.4: Comparison of mean convergence time between clustering methods with constant ω in discrete split case

After the convergence speed and the amount of required broadcast, another important metric for our algorithm is its clustering accuracy. Indeed, our goal is to compute average inside clusters so our algorithms must detect them accurately. Based on Figure 5.6, we can see that the tree first algorithms have a high accuracy. We note that the standard deviation algorithm is less accurate than the two others. Indeed, border nodes with this algorithm can leave their original cluster because of the communication between the two clusters. Finally, the local consensus algorithm exhibits an accuracy between 65% and 80%. This low accuracy confirm our expectations based on the fact that a border area is built by this algorithm to separate clusters.

This explication is confirmed by Figure 5.7. In this graph, we can observe that local consensus algorithm always builds at least one excessive cluster. The bad performance of this algorithm are then due to the initial value distribution which is discrete with an algorithm design for a continuous distribution.

It is also important to node that all these simulations were performed with parameter settings adapted to our toy scenario for the three first algorithms. Indeed, with a bad parameter setting, these algorithms can have worse performance. With this discrete

split scenario, it is especially true for the standard deviation algorithm that nodes, with a wrong parameter setting, should not detect they are border-node and then the algorithm will reach the AC over the whole network quickly.

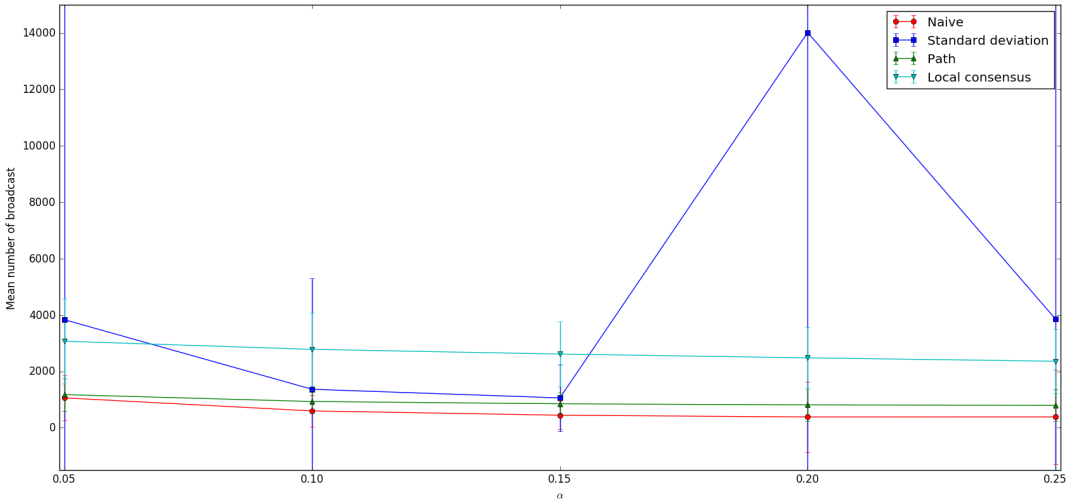


FIGURE 5.5: Comparison of mean number of broadcast between clustering methods with constant ω in discrete split case

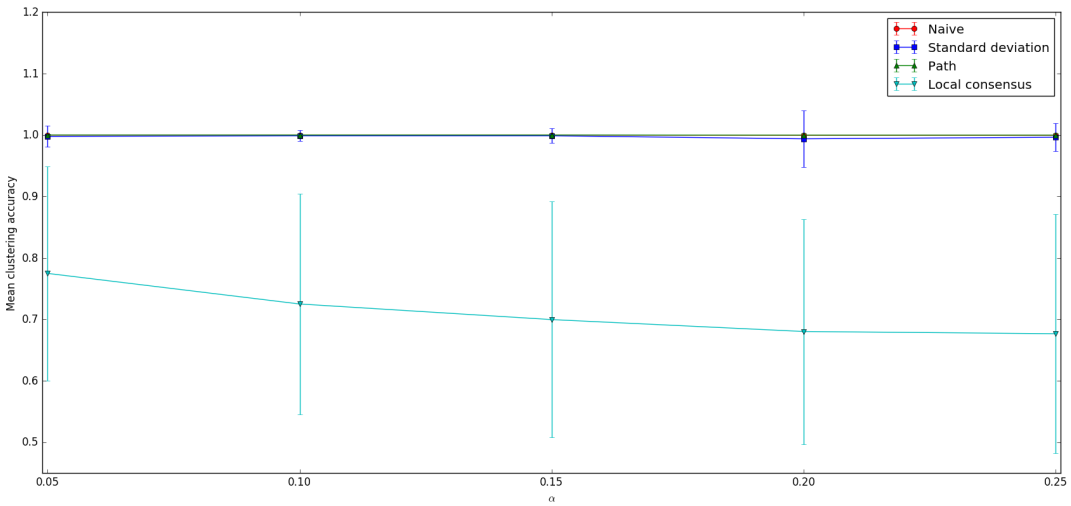


FIGURE 5.6: Comparison of mean cluster accuracy between clustering methods with constant ω in discrete split case

For this last algorithm, it is also interesting to remark that our way to define when consensus is reached plays in its favour. Indeed, the algorithm reaches the consensus but, as we explained it in the previous section, this algorithm will leave this consensus to reach AC over the whole network if it continues to run without corrective mechanism to maintain it.

As we say previously, the local consensus algorithm has been designed to work with a continuous value distribution where we can observe a transition area between clusters. We present here computer simulation results on this scenario for this algorithm and compare them with results obtained with the discrete split case.

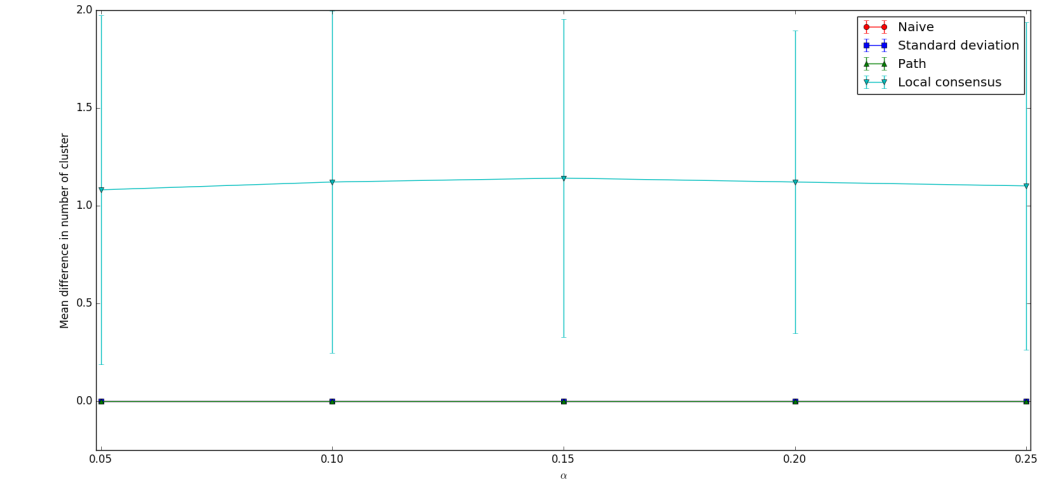


FIGURE 5.7: Comparison of mean difference in number of cluster between clustering methods with constant ω in discrete split case

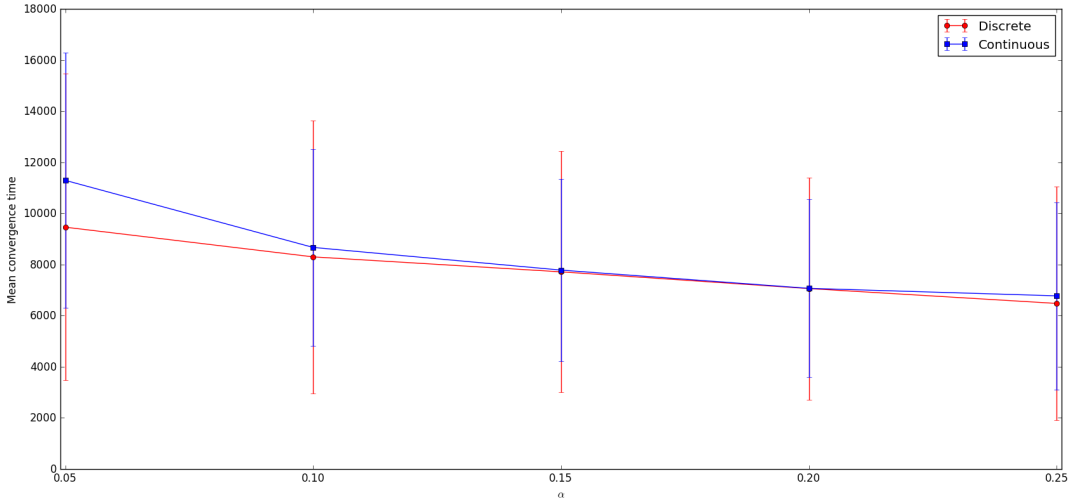


FIGURE 5.8: Comparison of local consensus algorithm mean convergence time with constant ω in discrete and continuous split case

First, there is almost no difference between the two scenarios in term of convergence speed according to Figure 5.8. The algorithm seems a bit slower with continuous split case but we consider, based on the error bars, that there is virtually no difference.

It is not the same for the mean number of broadcast for which we observe a significant difference in Figure 5.9. We explain this difference by the difference in the number of nodes that need to reach consensus. Indeed, border nodes from each cluster tends to leave their cluster to build a transition cluster between existing cluster in this algorithm but there is also nodes in the transition area that want to form this transition cluster in the continuous case. Based on this, we can conclude that more broadcast are required to reach consensus because more nodes need to reach it.

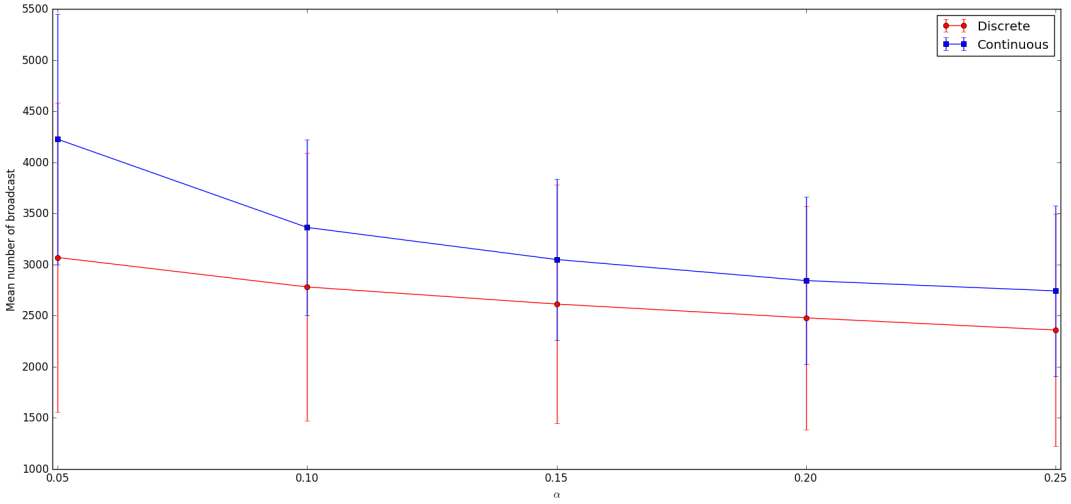


FIGURE 5.9: Comparison of local consensus algorithm mean number of broadcast with constant ω in discrete and continuous split case

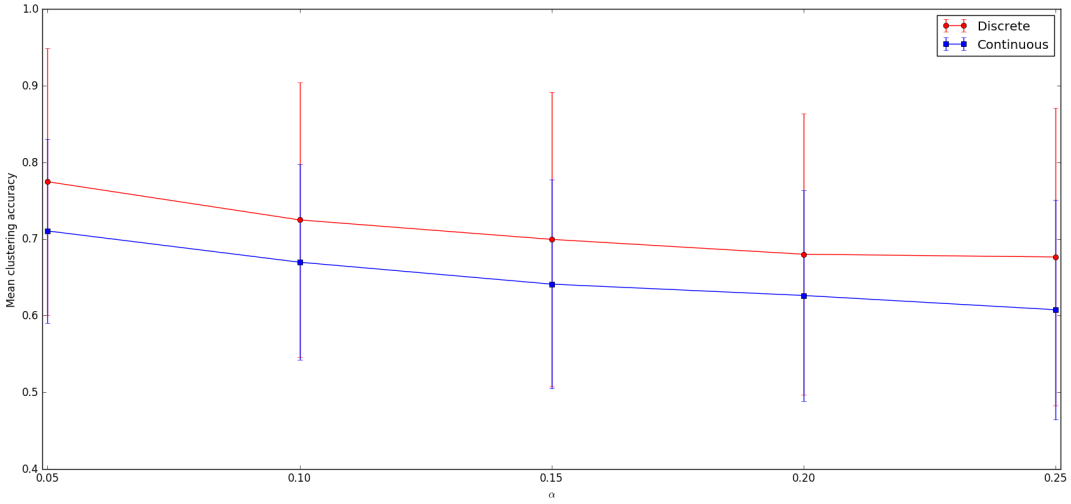


FIGURE 5.10: Comparison of local consensus algorithm mean clustering accuracy with constant ω in discrete and continuous split case

About clustering accuracy, we can observe a drop based on Figure 5.10. We conclude from this drop that our algorithm have a major issue with cluster borders detection. Even if it create less artificial clusters based on Figure 5.11, it seems that more nodes are not in the good cluster. This means that our algorithm is more accurate in the detection of the number of clusters but less accurate to select the good cluster for each node

On this point, our results do not meet our expectations but we can give a clue to explain this. For this, we make a parallel with set theory. Considering the network as a set, our algorithm tries to define two subsets corresponding to high value nodes and low value nodes that have reached consensus. Then, all nodes that do not fit one of those definitions is put in a third subset corresponding to the transition area. The problem is that our algorithm is not able to properly detect if nodes have reached

consensus or not. Indeed, using the standard deviation is a good measure to detect global consensus but some nodes that have effectively reached consensus have high standard deviation because some of their neighbours do not have reached it or are in another cluster. Those nodes, the border nodes in fact, are then almost always put in the wrong cluster. Furthermore, we assume that the number of border nodes increases in the continuous split scenario.

Another parameter that should be considered is the high variance of node values in the transition area. Because of the randomness of the algorithm, which is totally normal for stochastic process, we can encounter situations where the transition cluster will spontaneously split itself in multiple parts. Based on Figure 5.11, we can estimate this happens approximately half of the time.

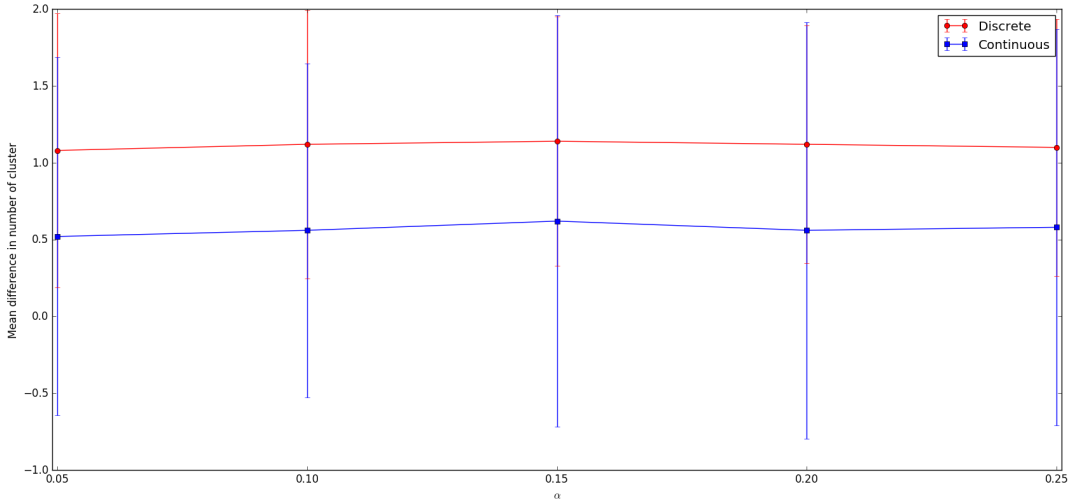


FIGURE 5.11: Comparison of local consensus algorithm mean difference in number of cluster with constant ω in discrete and continuous split case

Conclusion

With the growing of the Internet of Things and the beginning of the Internet of Materials, we expect a fast increase of connected device. We then need to rethink our way to communicate in order to avoid network congestion especially in high-density sensors networks.

In this work, we presented in Chapter 3 the AC algorithm with single update from Peper. This algorithm is a solution he proposed to respond to the problematic of high-density networks and the start of all this work. Indeed, this algorithm introduces a distributed solution to the average consensus problem using asymmetric broadcast communication allowing a very light communication protocol. For this algorithm, we principally focus on the impact of network topology since most of other results as been previously exposed in [12].

We then searched a way to improve this algorithm and proposed for this an AC algorithm with multiple updates that we presented in Chapter 4. For this algorithm, we first provide a numerical evaluation based on computer simulations as complete as possible. Then, we developed a mathematical model of it to serve as a basis for a mathematical analysis.

Finally, we decided in Chapter 5 to consider a more sophisticated scenario where we want to achieve consensus inside clusters instead over the whole graph. The goal of this was mainly to show some fancy possibilities of algorithms based on our works exposed in previous chapters. We proposed then a clustering consensus algorithm with different policies and make a first analysis of their performance.

To continue this work, some points in our analysis need a further investigation and could represent a good starting point. In particular, the link between p_l and λ for the AC algorithm with multiple updates is a key point to study in order to infirm or confirm our hypothesis that all parameters are linked to λ. Another possible interesting element could be to extend our analysis to bigger values of α. Indeed, we focus on this work on value of α behind 0.5 but we evoke the possibility of an optimal value for this parameter without prove it.

Other future works could be the development of algorithms to compute other statistical values based on our work for the AC algorithm. We think to the median which is an important statistical value in biological study and the standard deviation that we use a lot in this work.

Finally, the transposition of our algorithms to spiking sensor networks must be considered as a future works. Spiking sensors are similar to our nanosensors but they communicates using spikes. Peper present this type of sensor in [13] and explain the different ways communication can be implemented using spikes. We can also consider applications to binary sensors with noise where nodes can only say if they

measure something over or under a threshold value. Then, we use the fact that all node have a different threshold value because of the noise to estimate the real measure. Then, we could combine those two constraints on sensor which correspond to the objective of Peper.

If we had to make this work again, we should start our work on the average consensus algorithm with multiple updates by its mathematical model. Indeed, we use the balanced ω determined for the the algorithm with single update in our computer simulations because we do not have it to determine the optimal value of ω_i. This is of course an option to compare their performance but all other results are difficult to exploit given that a new way to compute the balanced ω for this algorithm can come from mathematical analysis.

Bibliography

[1] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Reviews of modern physics* 74.1 (2002), p. 47.

[2] Konstantin Avrachenkov, Mahmoud El Chamie, and Giovanni Neglia. "A local average consensus algorithm for wireless sensor networks". In: *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. IEEE. 2011, pp. 1–6.

[3] Stephen Boyd et al. "Randomized gossip algorithms". In: *IEEE/ACM Transactions on Networking (TON)* 14.SI (2006), pp. 2508–2530.

[4] Haisheng Chen et al. "Progress in electrical energy storage system: A critical review". In: *Progress in Natural Science* 19.3 (2009), pp. 291–312.

[5] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. "Energy-efficient communication protocol for wireless microsensor networks". In: *System sciences, 2000. Proceedings of the 33rd annual Hawaii international conference on*. IEEE. 2000, 10–pp.

[6] Kenji Leibnitz et al. "Maximum entropy based randomized routing in data-centric networks". In: *Network Operations and Management Symposium (APNOMS), 2013 15th Asia-Pacific*. IEEE. 2013, pp. 1–6.

[7] James Macaulay, Lauren Buckalew, and Gina Chung. "Internet of things in logistics". In: *DHL Trend Research & Cisco Consulting Services, DHL Customer Solutions & Innovation* 53844 (2015).

[8] Loreto Mateu and Francesc Moll. "Review of energy harvesting techniques and applications for microelectronics (Keynote Address)". In: *Microtechnologies for the New Millennium 2005*. International Society for Optics and Photonics. 2005, pp. 359–373.

[9] Michael Andrew McEvoy and Nikolaus Correll. "Materials that couple sensing, actuation, computation, and communication". In: *Science* 347.6228 (2015), p. 1261689.

[10] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. "Consensus and cooperation in networked multi-agent systems". In: *Proceedings of the IEEE* 95.1 (2007), pp. 215–233.

[11] Travis E Oliphant. "Python for scientific computing". In: *Computing in Science & Engineering* 9.3 (2007).

[12] Ferdinand Peper et al. "Average consensus in asymmetric broadcasting wireless sensor networks through gossiping". In: *Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services*. ACM. 2016, pp. 171–176.

[13] Ferdinand Peper et al. "Low-complexity nanosensor networking through spike-encoded signaling". In: *IEEE Internet of Things Journal* 3.1 (2016), pp. 49–58.

[14] Dongjin Seo et al. "Model validation of untethered, ultrasonic neural dust motes for cortical recording". In: *Journal of neuroscience methods* 244 (2015), pp. 114–122.

[15] Dongjin Seo et al. "Neural dust: An ultrasonic, low power solution for chronic brain-machine interfaces". In: *arXiv preprint arXiv:1307.2196* (2013).

[16] John Nikolas Tsitsiklis. *Problems in Decentralized Decision making and Computation.* Tech. rep. DTIC Document, 1984.

[17] Duncan J Watts and Steven H Strogatz. "Collective dynamics of 'small-world'networks". In: *nature* 393.6684 (1998), pp. 440–442.

[18] Lin Xiao and Stephen Boyd. "Fast linear iterations for distributed averaging". In: *Systems & Control Letters* 53.1 (2004), pp. 65–78.

[19] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. "Distributed average consensus with least-mean-square deviation". In: *Journal of Parallel and Distributed Computing* 67.1 (2007), pp. 33–46.

[20] Lin Xiao, Stephen Boyd, and Sanjay Lall. "A scheme for robust distributed sensor fusion based on average consensus". In: *Proceedings of the 4th international symposium on Information processing in sensor networks*. IEEE Press. 2005, p. 9.

[21] Lin Xiao, Stephen Boyd, and Sanjay Lall. "A space-time diffusion scheme for peer-to-peer least-squares estimation". In: *Proceedings of the 5th international conference on Information processing in sensor networks*. ACM. 2006, pp. 168–176.

[22] Hua Yang, Fengji Ye, and Biplab Sikdar. "A dynamic query-tree energy balancing protocol for sensor networks". In: *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*. Vol. 3. IEEE. 2004, pp. 1715–1720.